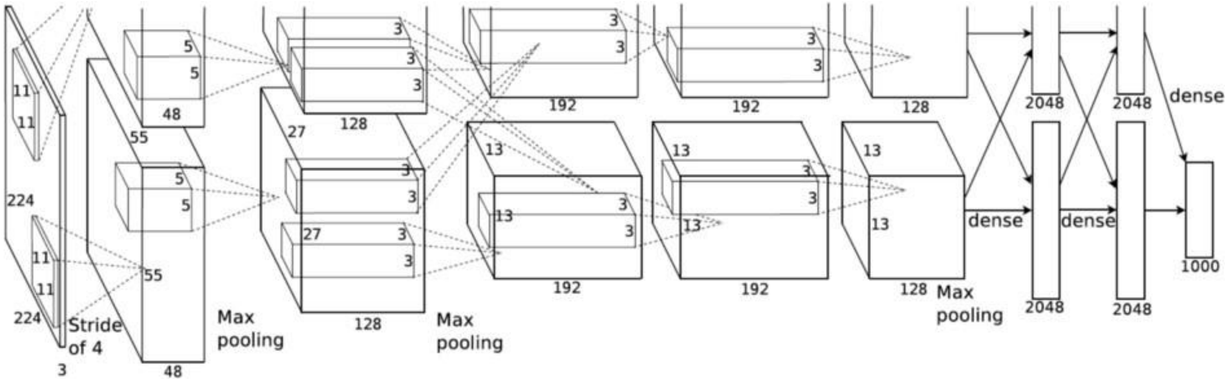


COMP 9517 Computer Vision

Deep Learning for Computer Vision

Challenges in CV

Consider object detection as an example:

- Variations in viewpoint
- Differences in illumination
- Hidden parts of images
- Background clutter

Content: link

Computer Vision and Deep Learning

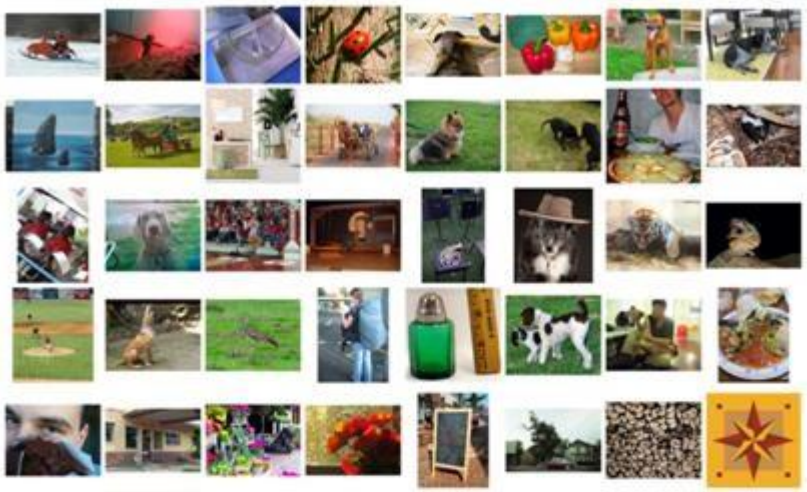
- The earliest research in computer vision started way back in 1950s. Since then, we have come a long way but still find ourselves far from the ultimate objective.
- But with neural networks and deep learning, we have become empowered like never before.
- Successful applications of deep learning in CV include
  - Object recognition – ImageNet
  - Context Analysis
  - Semantic Segmentation
  - Classification
  - Retrieval
  - Self-driving cars

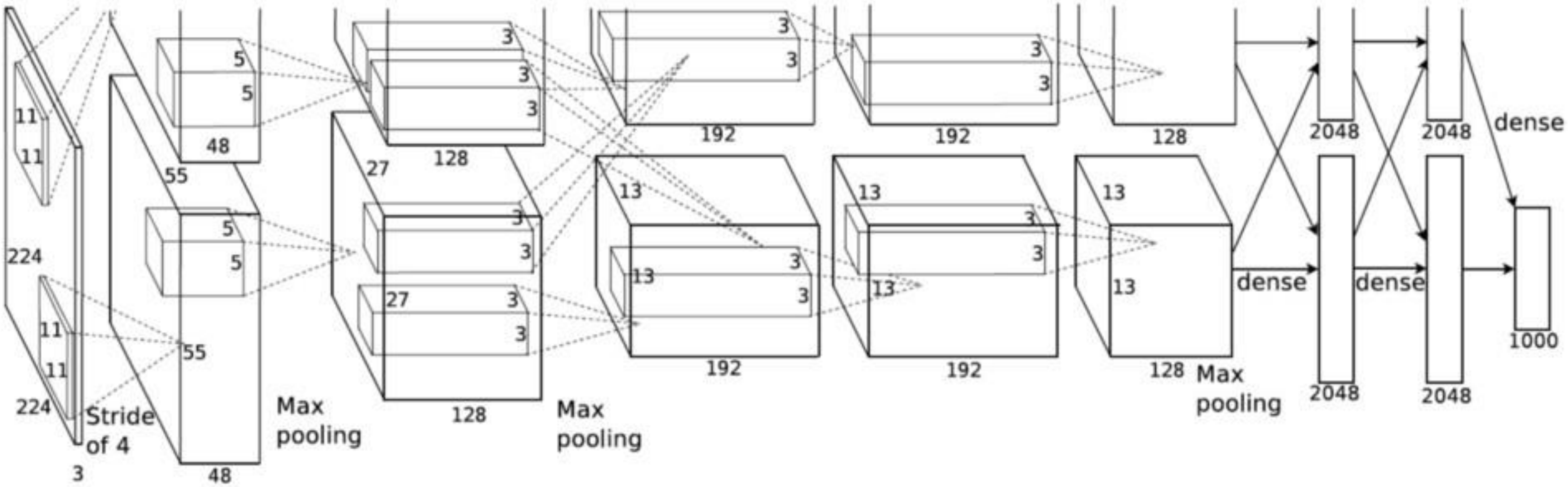
A bit of History

- Earliest studies about visual mechanics of animals by Hubel and Weisel emphasised the importance of edge detection for solving CV problems
    - Early processing in cat visual cortex looks like it is performing convolutions that are looking for oriented edges and blobs
    - Certain cells are looking for edges with a particular orientation at a particular spatial location in the visual field
    - This inspired convolutional neural networks but was limited by the lack of computational power
- Hinton et al. reinvigorated research into deep learning and proposed a greedy layer wise training technique
    - In 2012, Alex Krizhevesky et al. won ImageNet challenge and proposed the well recognised AlexNet Convolutional Neural Network

A bit of History

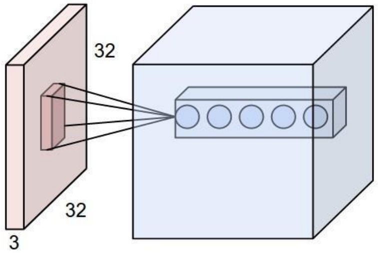
- AlexNet (2012)
  - ImageNet classification
  - Images showing 1000 object categories





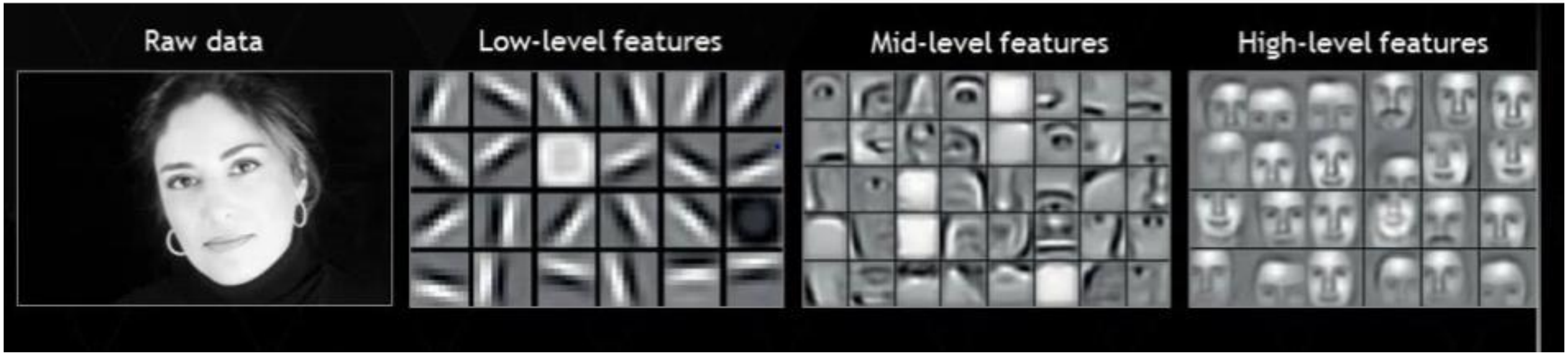
Deep Learning

- Deep learning is a collection of artificial neural network techniques that are widely used at present
- Predominantly, deep learning techniques rely on large amounts of data and deeper learning architectures
- Some well known paradigms:
  - **Convolutional Neural Networks (CNNs)**
  - Recurrent Neural Networks
  - Auto-encoders
  - Restricted Boltzmann Machines



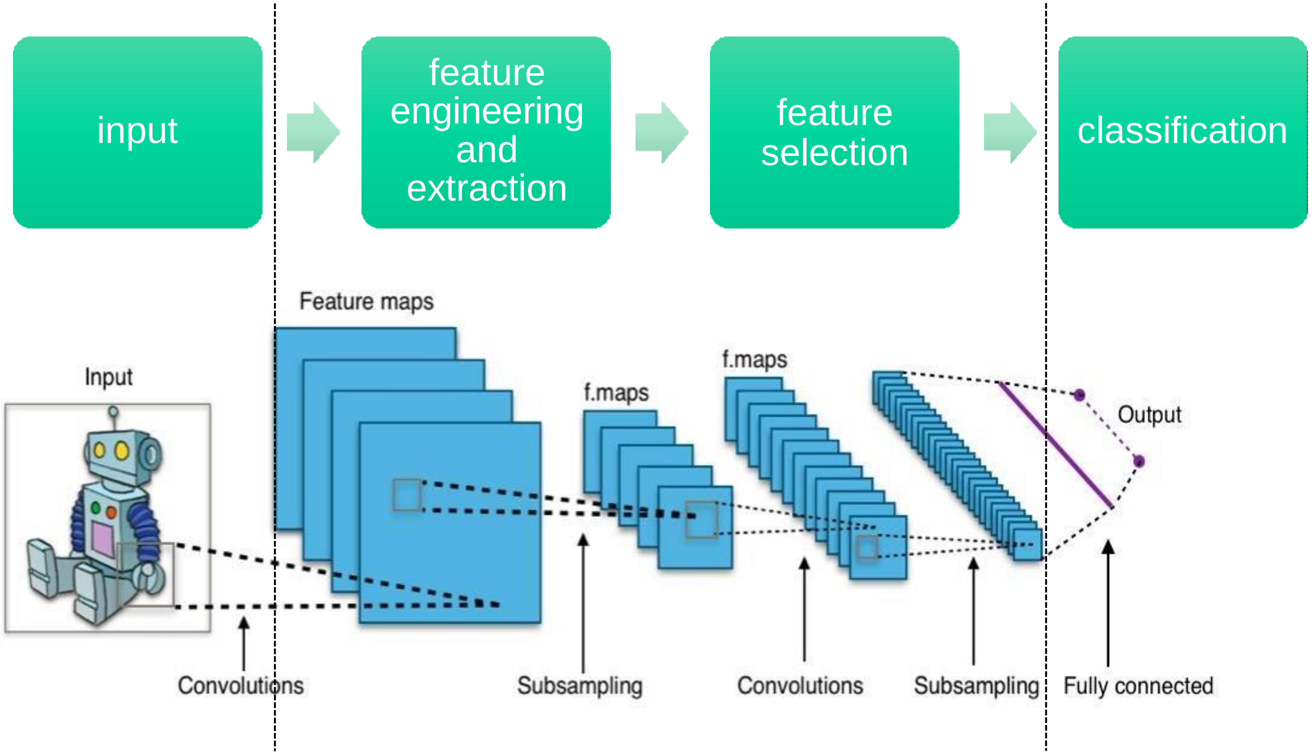
Traditional Approach vs DL

- CNNs can be interpreted as gradually transforming the images into a representation in which the classes are separable by a linear classifier.
- CNNs will try to learn low-level features such as edges and lines in early layers, then parts of objects and then high-level representation of an object in subsequent layers.



https://www.analyticsvidhya.com/blog/2017/04/comparison-between-deep-learning-machine-learning/

Traditional Approach vs DL



input → feature engineering and extraction → feature selection → classification

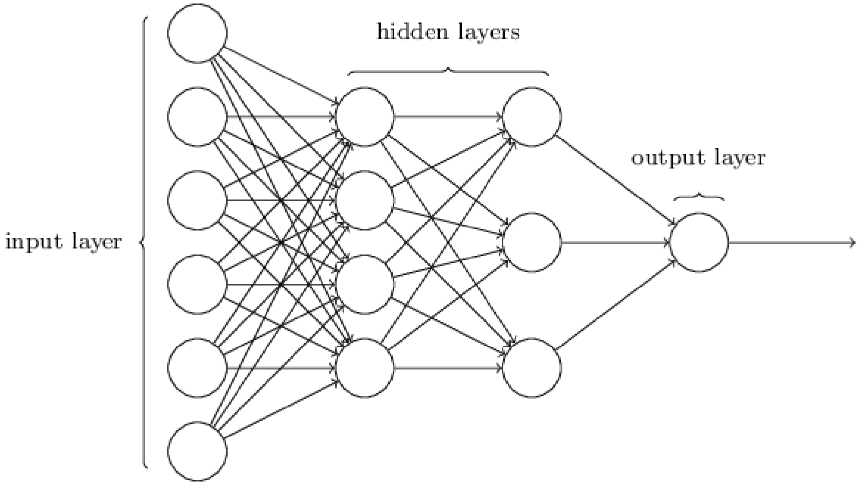
https://towardsdatascience.com/convolutional-neural-networks-for-all-part-i-cdd282ee7947

CNNs

- CNNs are very similar to regular Neural Networks
    - Made up of neurons with learnable weights
- CNN architecture assumes that inputs are images
    - So that we have local features
    - Which allows us to
        - encode certain properties in the architecture that makes the forward pass more efficient and
        - significantly reduces the number of parameters needed for the network

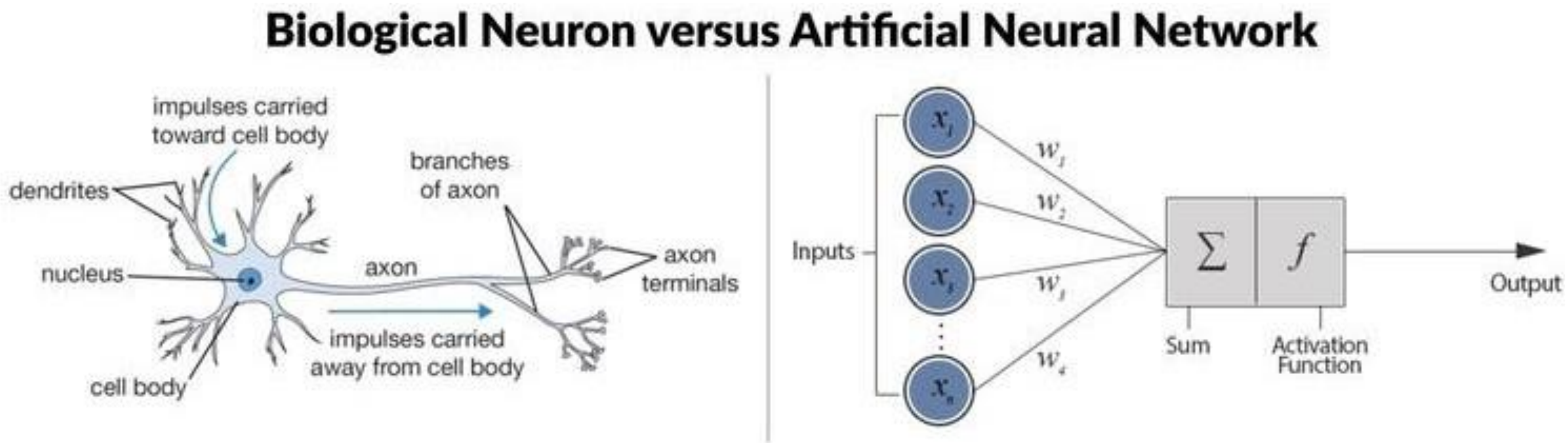
Neural Networks

- Artificial Neural Networks are inspired by human nervous system
- NNs are composed of a large number of interconnected processing elements known as neurons
- They use supervised error correcting rules with back-propagation to learn a specific task



http://statsmaths.github.io/stat665/lectures/lec12/lecture12.pdf

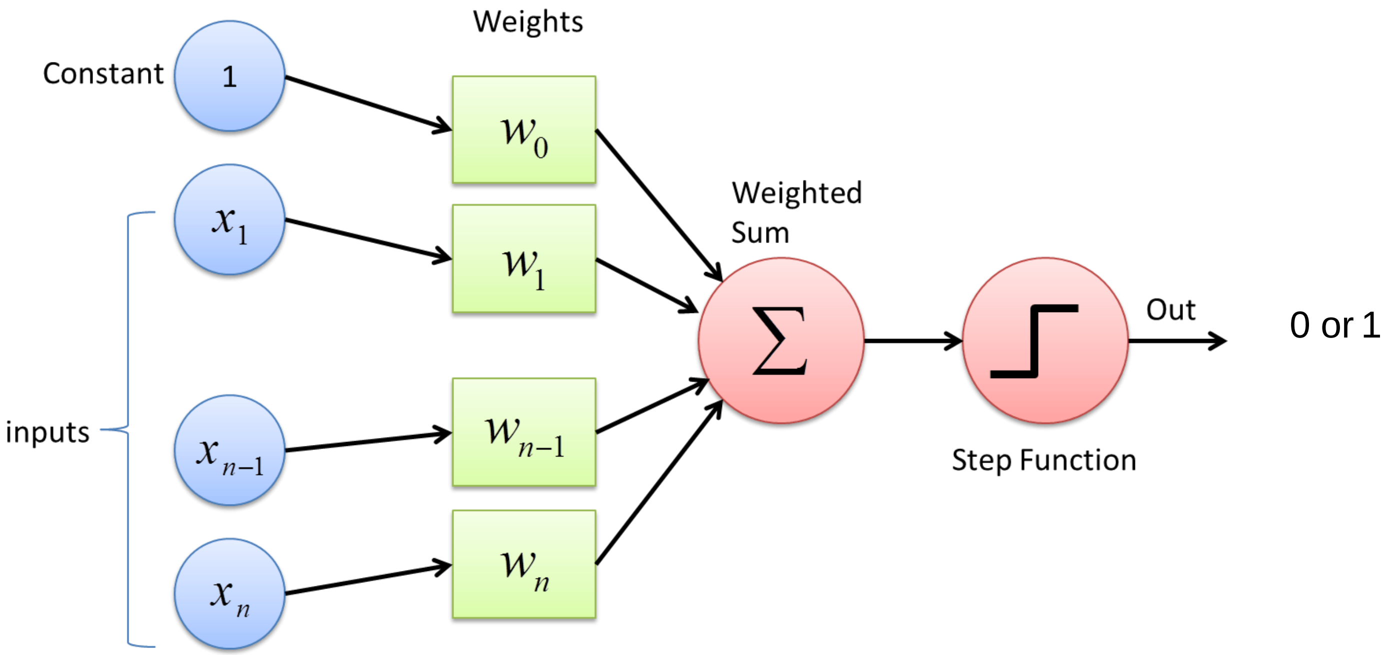
NN: Perceptron



Each neuron has multiple dendrites and a single axon. The neuron receives its inputs from its dendrites and transmits its output through its axon. Both inputs and outputs take the form of electrical impulses. The neuron sums up its inputs, and if the total electrical impulse strength exceeds the neuron's firing threshold, the neuron fires off a new impulse along its single axon. The axon, in turn, distributes the signal along its branching synapses which collectively reach thousands of neighboring neurons.

https://towardsdatascience.com/from-fiction-to-reality-a-beginners-guide-to-artificial-neural-networks-d0411777571b

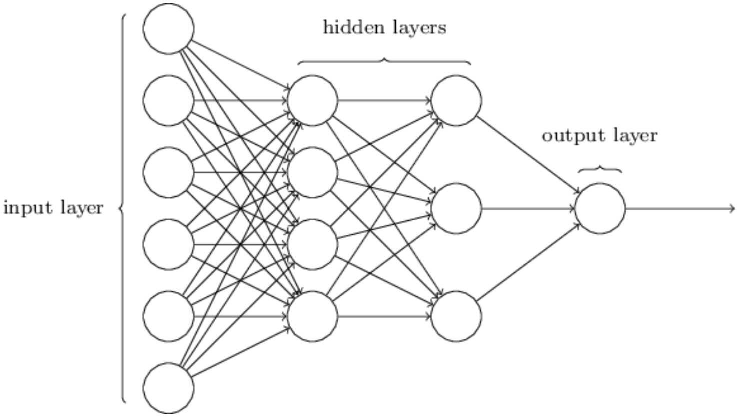
NN: Perceptron

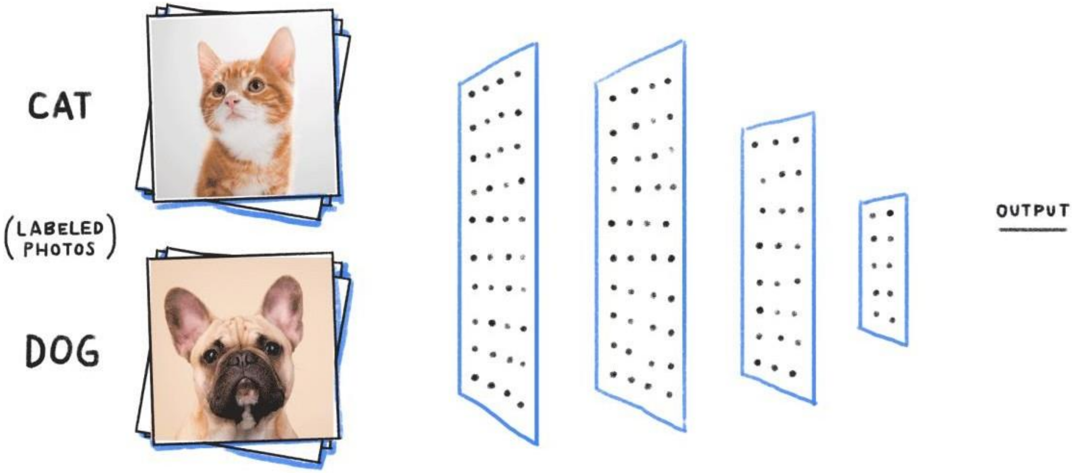


Perceptron is a single-layer neural network
A multi-layer perceptron becomes Neural Networks

https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

Neural Network Architecture





https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

Why CNNs?

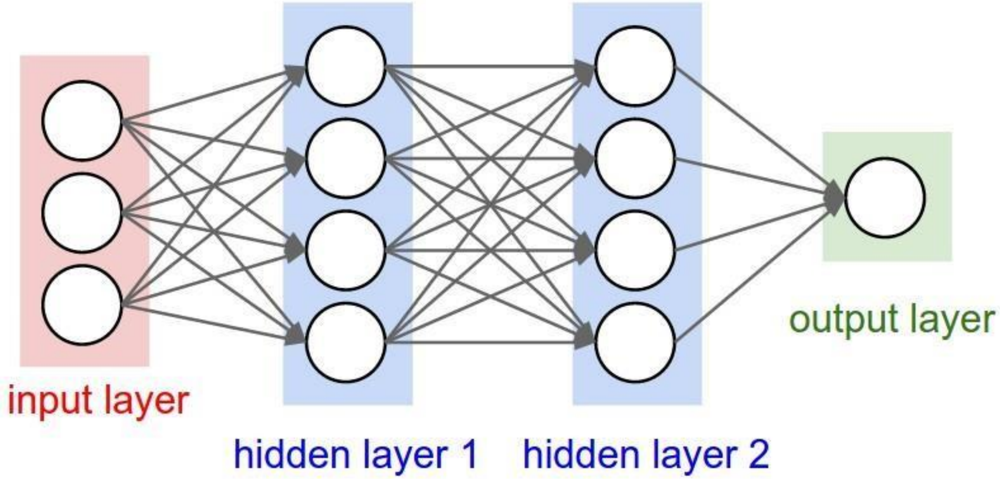
- The problem with regular NNs is that they do not scale well with dimensions (i.e. larger images)
  - Eg: 32x32 image with 3 channels (RGB) – a neuron in first hidden layer would have 32x32x3 = 3,072 weights : manageable.
  - Eg: 200x200 image with 3 channels – a neuron in first hidden layer would have 200x200x3 = 120,000 weights and we need at least several of these neurons which makes the weights explode.

So what is different?

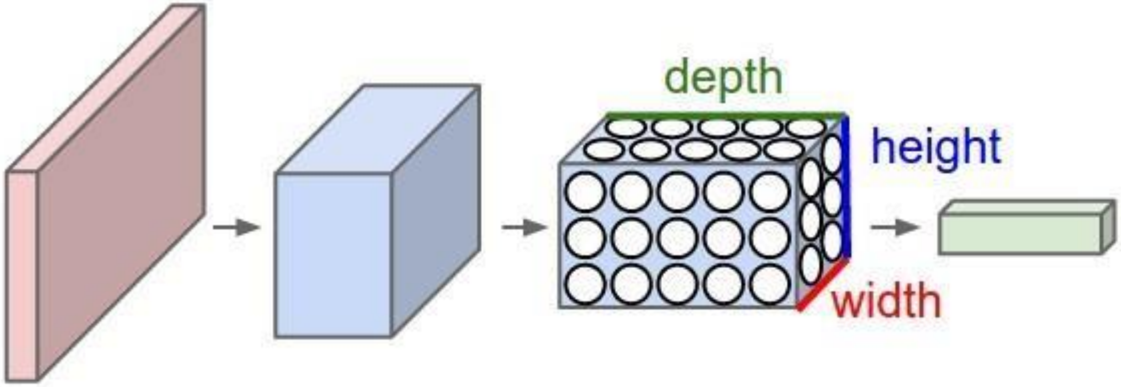
- In contrast, CNNs consider 3-D volumes of neurons and propose a parameter sharing scheme that minimises the number of parameters required by the network.
- CNN neurons are arranged in 3 dimensions: Width, Height and Depth.
- Neurons in a layer are only connected to a small region of the layer before it (hence not fully connected)

So what is different?

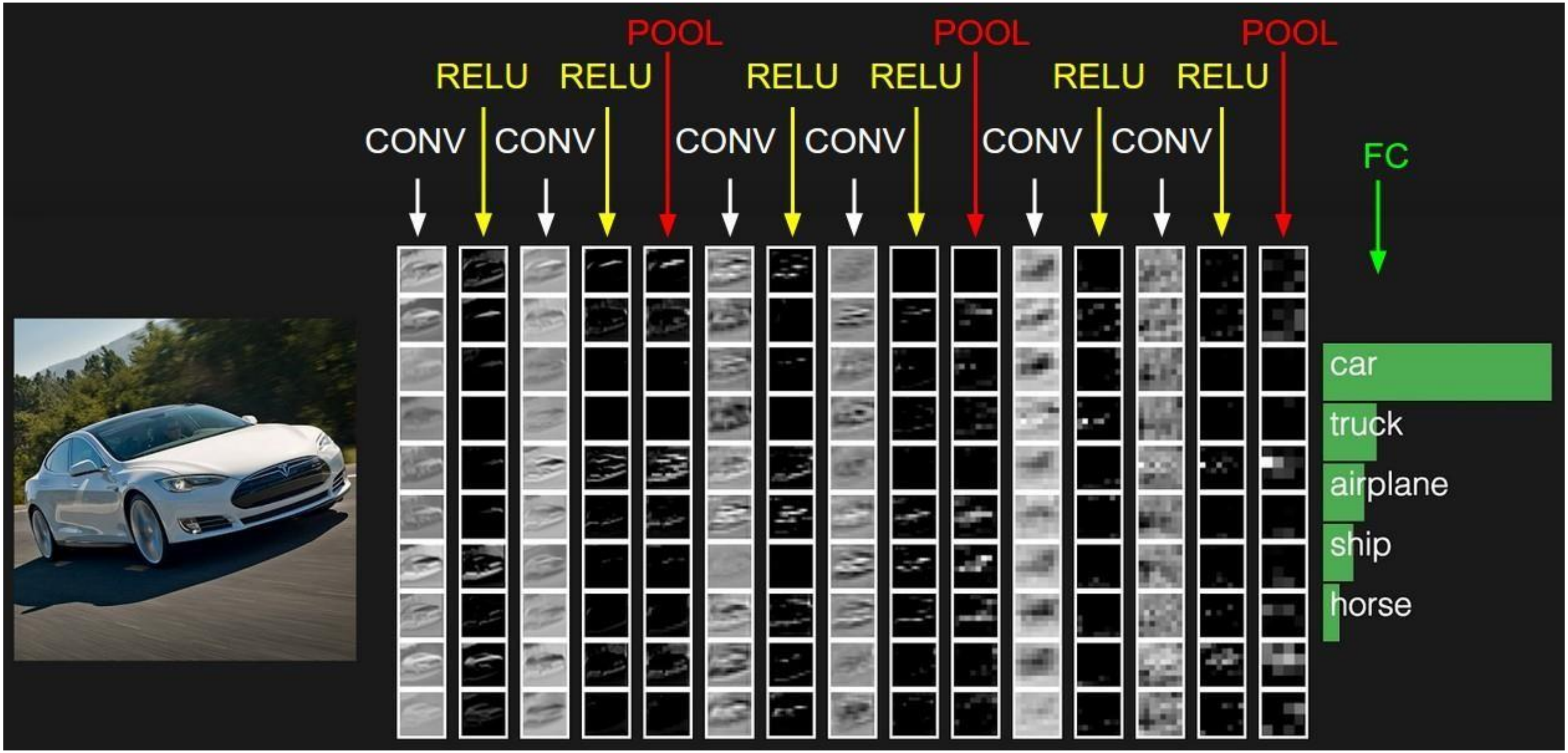
NN



CNN

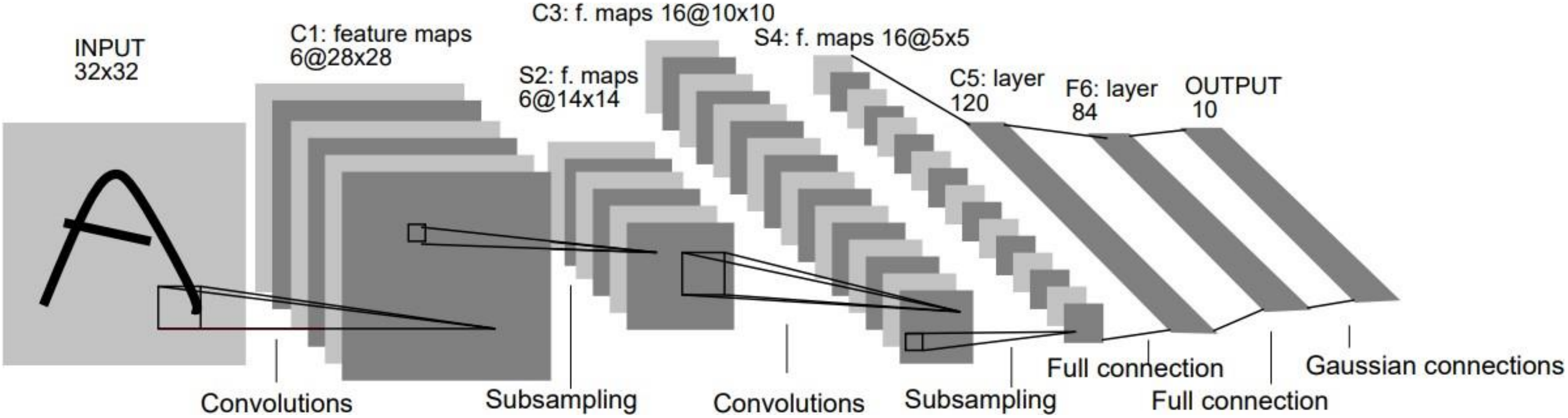


CNN Architecture



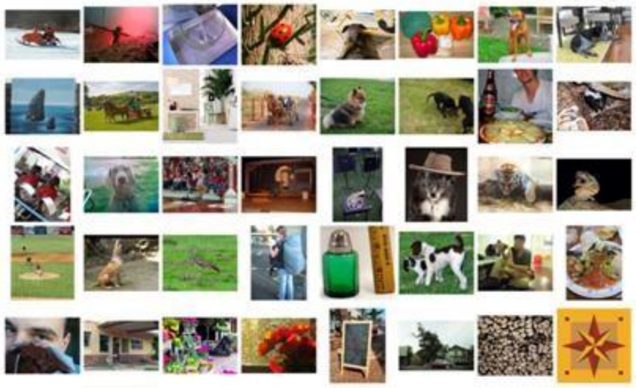
CNN Architecture

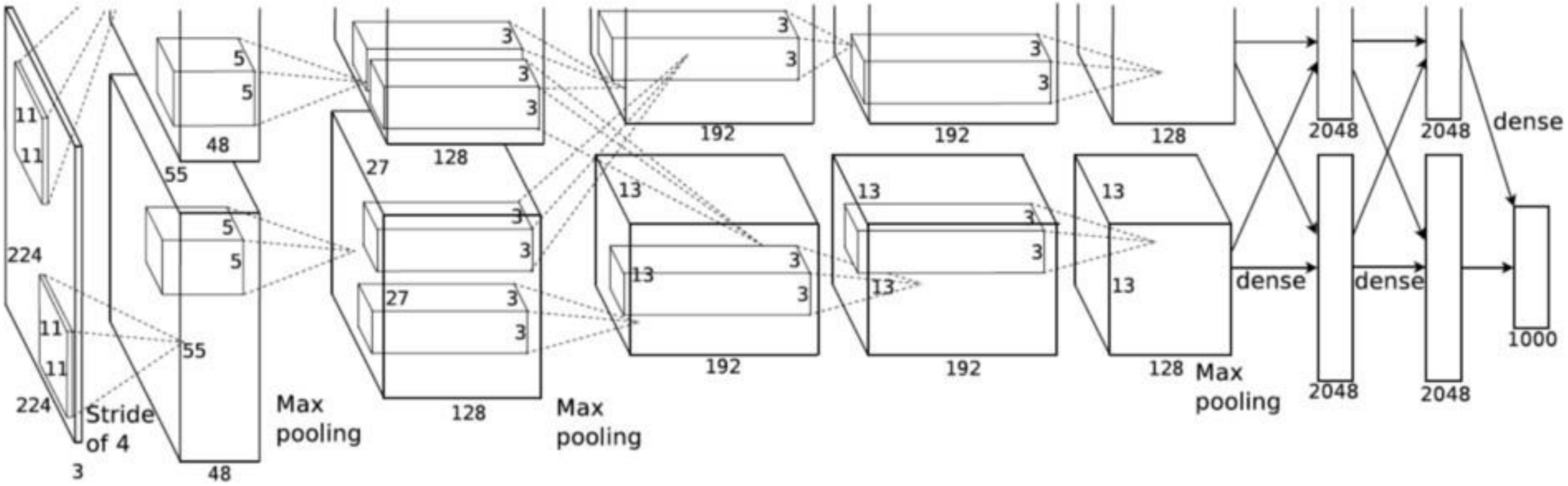
- LeNet-5
  - The very first CNN
  - Handwritten digit recognition



CNN Architecture

- AlexNet
  - The start of booming of deep learning
  - ImageNet classification



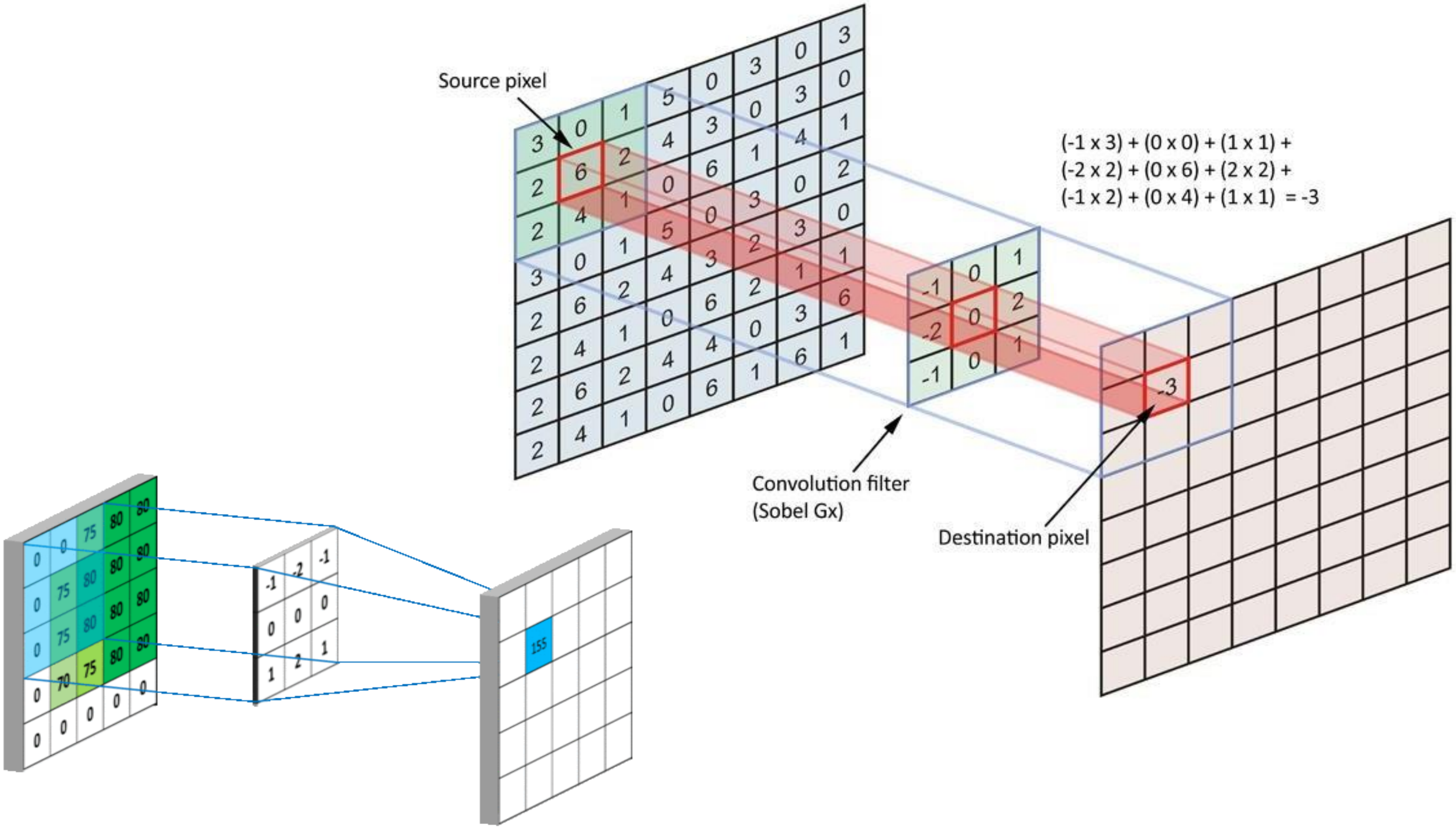


CNN Architecture

- CNNs are usually made up of various types of layers
  - Convolution Layer
  - Pooling Layer
  - ReLU Layer, and
  - Fully-Connected Layer
  - Dropout layer
  - Output layer
- This lecture provides a fundamental understanding of CNN
- Most recent CNN architectures have more different types of layers or models, which are out of the scope of this lecture

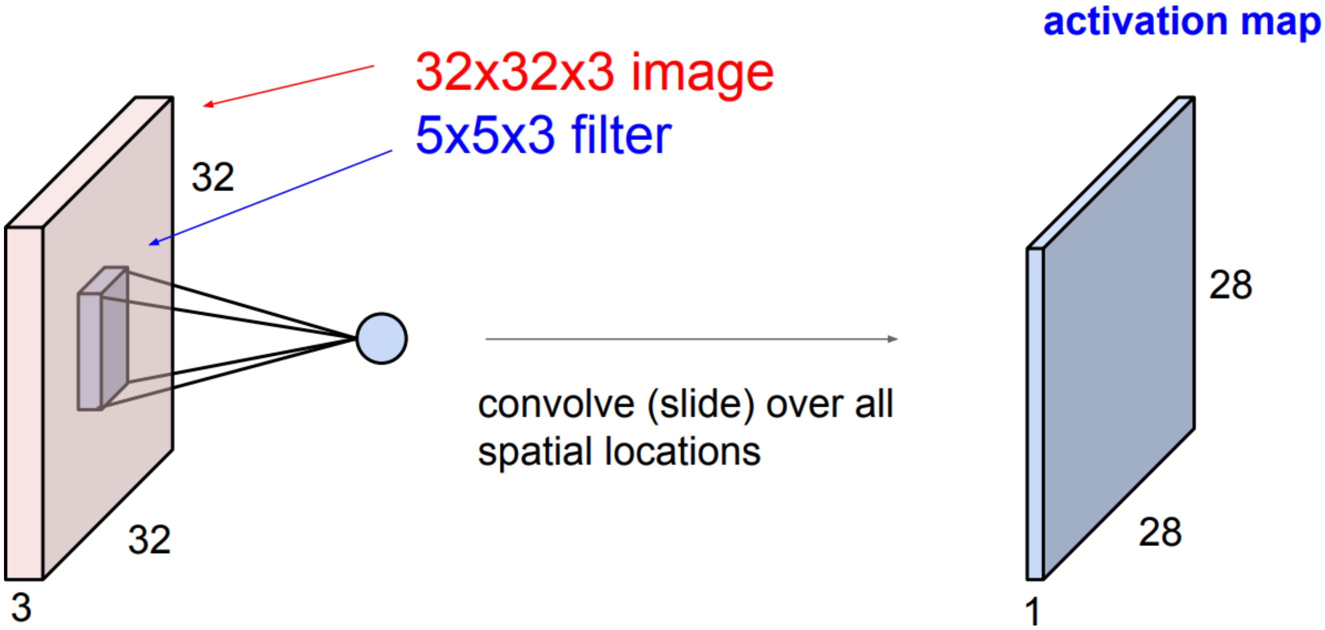
CNN: Convolutional Layer

Convolution



(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3

CNN: Convolutional Layer



32x32x3 image

5x5x3 filter

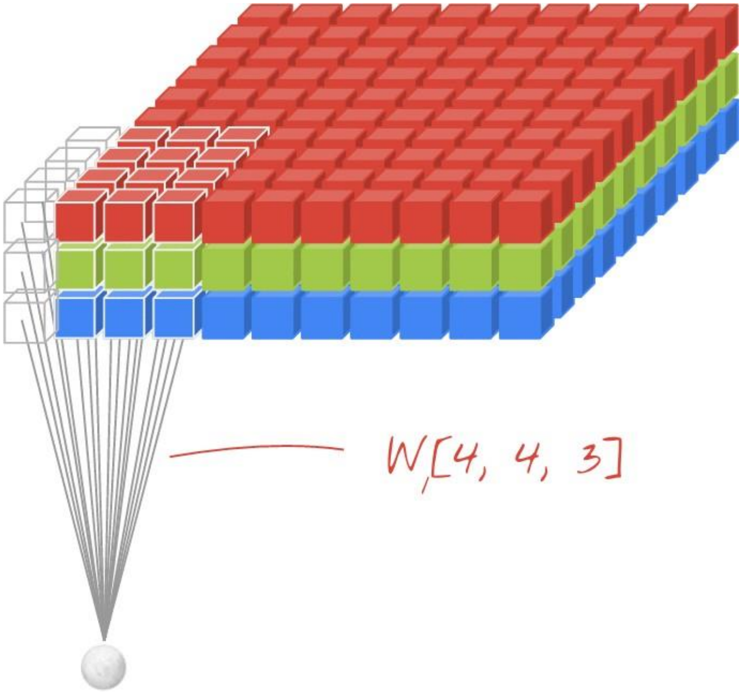
convolve (slide) over all spatial locations

activation map

CNN: Convolutional Layer

- The output of the Conv layer can be interpreted as holding neurons arranged in a 3D volume.
- The Conv layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume.
- During the forward pass, each filter is slid (convolved) across the width and height of the input volume, producing a 2-dimensional activation map of that filter.
- Network will learn filters (via backpropagation) that activate when they see some specific type of feature at some spatial position in the input.

CNN: Convolutional Layer



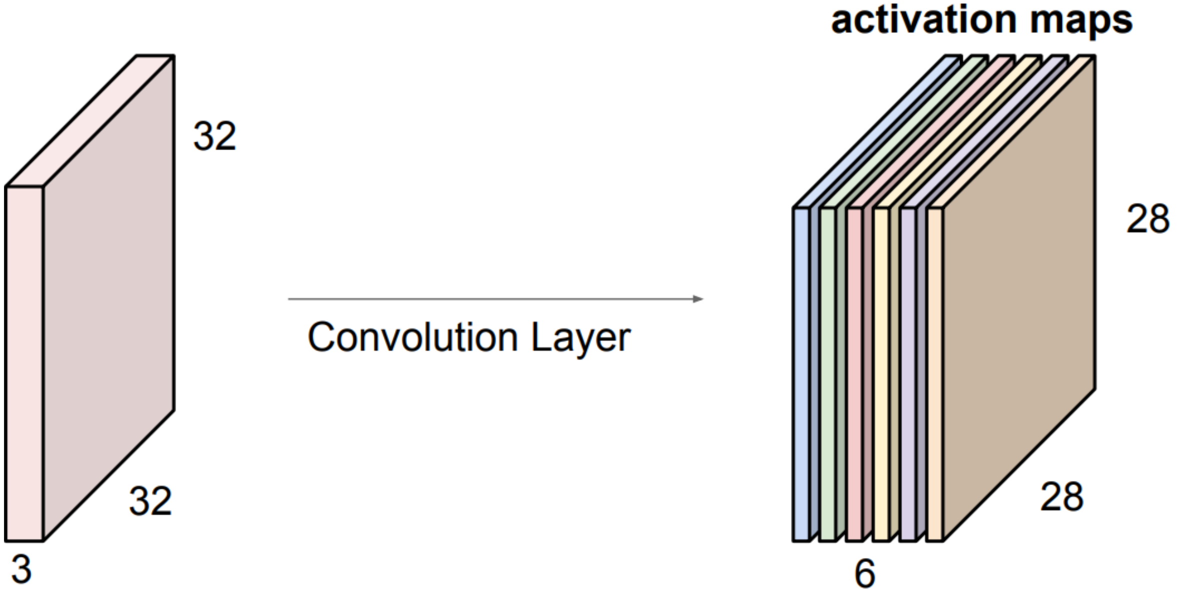
Original LINK

CNN: Convolutional Layer

- Stacking these activation maps for all filters along the depth dimension forms the full output volume
- Every entry in the output volume can thus also be interpreted as an output of a neuron that looks at only a small region in the input and ***shares parameters*** with neurons in the same activation map (since these numbers all result from applying the same filter)

CNN: Convolutional Layer

With 6 filters, we get 6 activation maps



CNN: Convolutional Layer
Local Connectivity

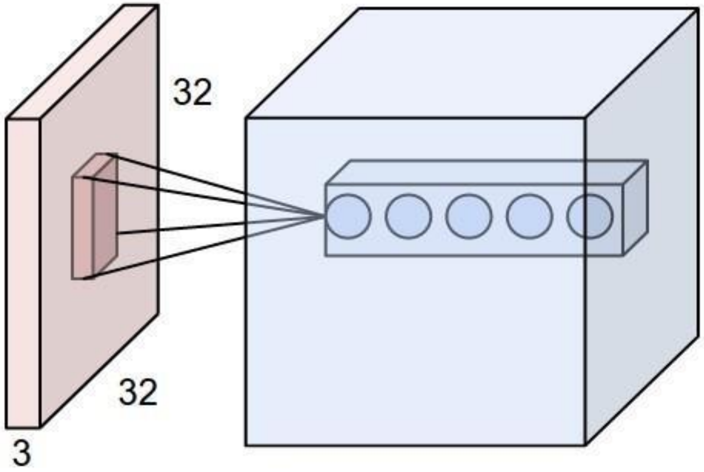
- As we have realized by now, it is impractical to use fully connected networks when dealing with high dimensional images/data
- Hence the concept of local connectivity: each neuron only connects to a local region of the input volume.
- The spatial extent of this connectivity is a hyperparameter called ***receptive field*** of the neuron.
- The extent of the connectivity along the depth axis is always equal to the depth of the input volume.
- The connections are local in space (along width and height), but always full along the entire depth of the input volume.

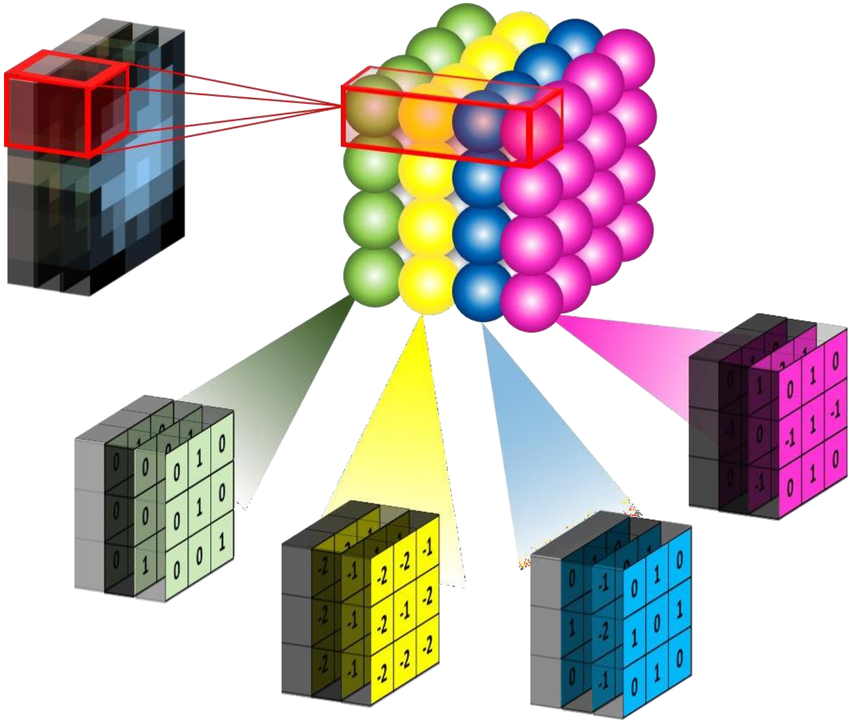
CNN: Convolutional Layer
Local Connectivity

- Eg1: Suppose that the input volume has size [32x32x3]. If the receptive field is of size 5x5, then each neuron in the Conv Layer will have weights to a [5x5x3] region in the input volume, for a total of 5*5*3 = 75 weights. Notice that the extent of the connectivity along the depth axis must be 3, since this is the depth of the input volume.
- Eg 2: Suppose an input volume had size [16x16x20], i.e. . Then using an example receptive field size of 3x3, every neuron in the Conv Layer would now have a total of 3*3*20 = 180 connections to the input volume. Notice that, again, the connectivity is local in space (e.g. 3x3), but full along the input depth (20).

CNN: Convolutional Layer

Local Connectivity

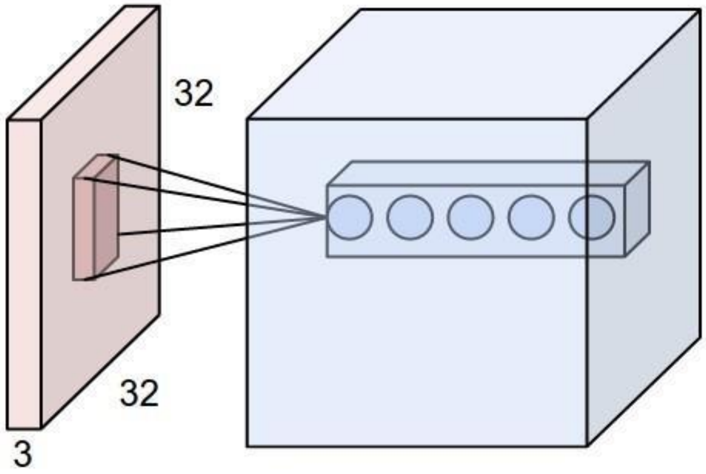




Original LINK

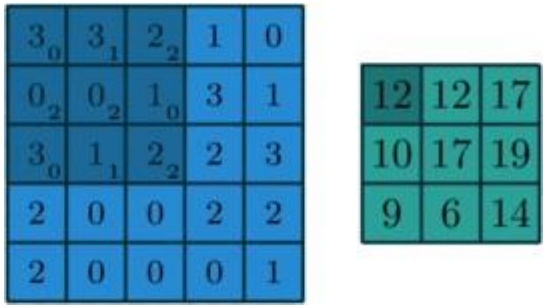
CNN: Convolutional Layer
Spatial Arrangement

- Three hyperparameters control the size of the output volume: the ***depth, stride*** and ***zero-padding***
  - ***Depth*** controls the number of neurons in the Conv layer that connect to the same region of the input volume



CNN: Convolutional Layer
Spatial Arrangement

- Three hyperparameters control the size of the output volume: the ***depth, stride*** and ***zero-padding***
  - ***Stride*** is the distance that the filter is moved by in spatial dimensions



3_0	3_1	2_2	1	0
0_2	0_2	1_0	3	1
3_0	1_1	2_2	2	3
2	0	0	2	2
2	0	0	0	1

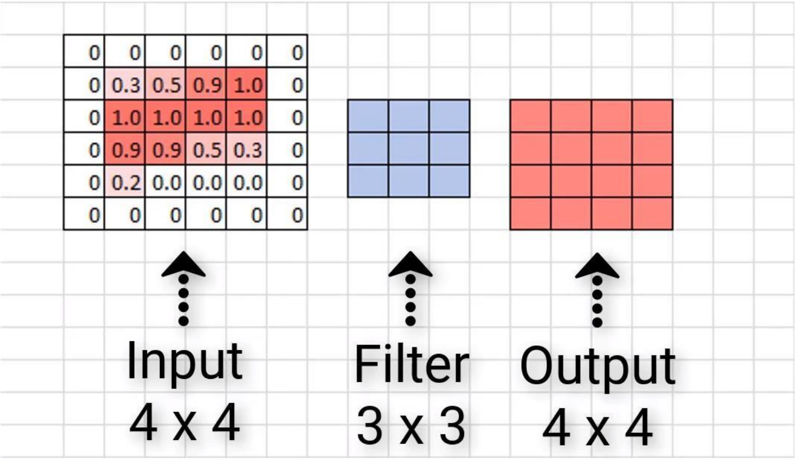
12	12	17
10	17	19
9	6	14

http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html

CNN: Convolutional Layer

Spatial Arrangement

- Three hyperparameters control the size of the output volume: the ***depth, stride*** and ***zero-padding***
  - ***Zero-padding*** is padding of the input with zeros spatially on the border of the input volume



0	0	0	0	0	0
0	0.3	0.5	0.9	1.0	0
0	1.0	1.0	1.0	1.0	0
0	0.9	0.9	0.5	0.3	0
0	0.2	0.0	0.0	0.0	0
0	0	0	0	0	0

https://deeplizard.com/learn/video/qSTv_m-KFk0

CNN: Convolutional Layer
Spatial Arrangement

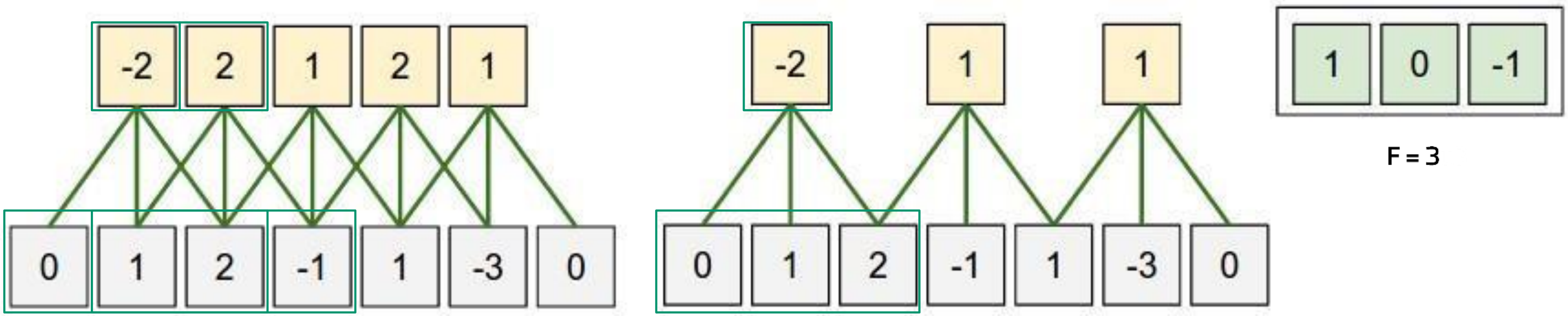
- We can compute the spatial size of the output volume as a function of the input volume size (W), the receptive field size of the Conv Layer neurons (F), the stride with which they are applied (S), and the amount of zero padding used (P) on the border:

$$(W-F+2P)/S+1$$

- If this number is not an integer, then the strides are set incorrectly and the neurons cannot be tiled so that they "fit" across the input volume neatly, in a symmetric way.

CNN: Convolutional Layer

Spatial Arrangement



W = 5 (Input Size)

P = 1 (zero-padding level 1)

S = 1

(5 – 3 + 2 * 1) / 1 + 1 = 5

W = 5 (Input Size)

P = 1 (zero-padding level 1)

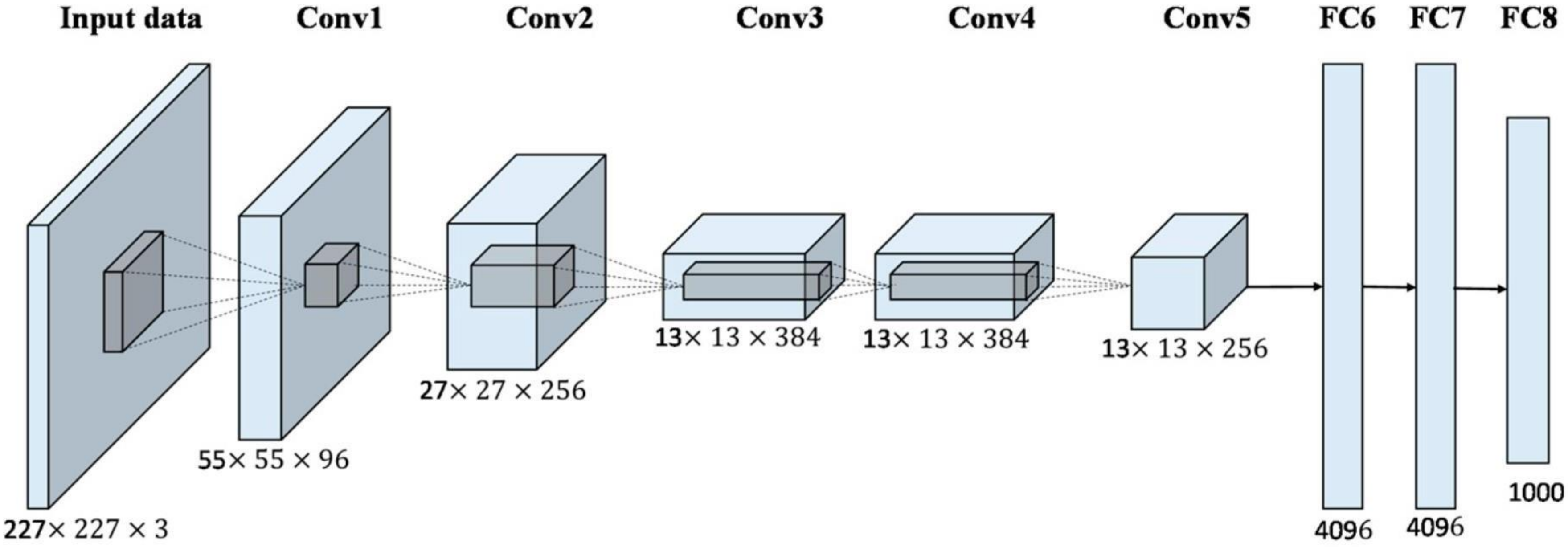
S = 2

(5 – 3 + 2 * 1) / 2 + 1 = 3

CNN: Convolutional Layer
Spatial Arrangement

- Real-world example. The AlexNet that won the ImageNet challenge in 2012 accepted images of size 227x227x3
- On the first Convolutional Layer, it used neurons with receptive field size F=11, stride S=4 and no zero padding P=0.
- Since (227 - 11)/4 + 1 = 55, and since the Conv layer had a depth of K=96, the Conv layer output volume had size 55x55x96
- Each of the 55*55*96 neurons in this volume was connected to a region of size 11x11x3 in the input volume
- Moreover, all 96 neurons in each depth column are connected to the same 11x11x3 region of the input, but of course with different weights

CNN: Convolutional Layer
Spatial Arrangement



CNN: Convolutional Layer
Parameter Sharing

- Parameter sharing scheme used in Convolutional Layers to control the number of parameters
  - In other words, denoting a single 2-D slice as a depth slice (e.g. a volume of size [55x55x96] has 96 depth slices, each of size [55x55]), we are going to constrain the neurons in each depth slice to use the same weights and bias
  - This is exactly what we do with spatial filters for signals/images!
- Motivation of parameter sharing
  - If one patch feature is useful to compute at some spatial position (x,y), then it should also be useful to compute at a different position (x2,y2).

CNN: Convolutional Layer
Parameter Sharing

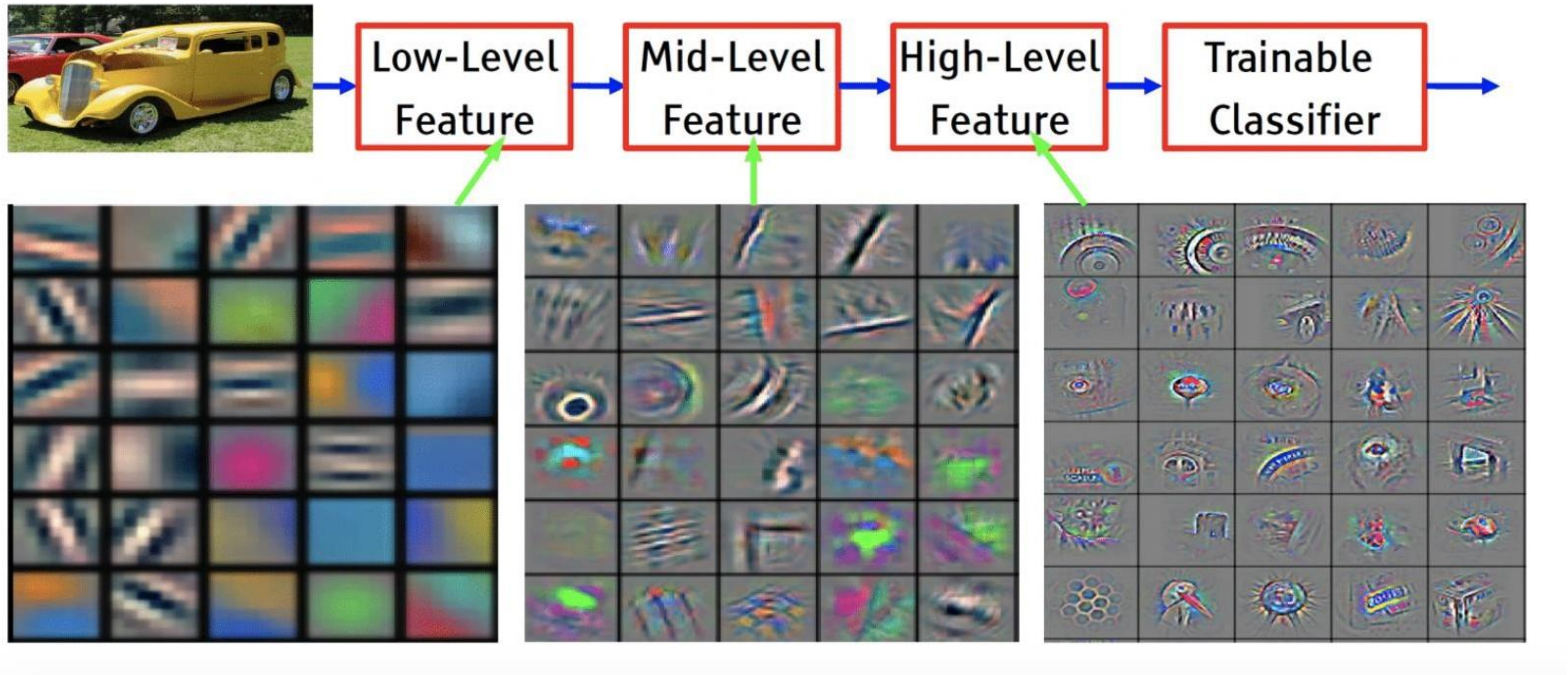
- Example:
  - In the AlexNet example, without parameter sharing, there are 55*55*96 = 290,400 neurons in the first Conv Layer, and each has 11*11*3 = 363 weights and 1 bias.
  - Together, this adds up to 290400 * 364 = 105,705,600 parameters on the first layer of the ConvNet alone. Clearly, this number is very high.
  - With this parameter sharing scheme, the first Conv Layer in our example would now have only 96 unique sets of weights (one for each depth slice), for a total of 96*11*11*3 = 34,848 unique weights, or 34,944 parameters (+96 biases).
  - Alternatively, all 55*55 neurons in each depth slice will now be using the same parameters.

CNN: Convolutional Layer
Parameter Sharing

- In practice during backpropagation, every neuron in the volume will compute the gradient for its weights, but these gradients will be added up across each depth slice and only update a single set of weights per slice.
- If all neurons in a single depth slice are using the same weight vector, then the forward pass of the Conv layer can in each depth slice be computed as a **convolution** of the neuron's weights with the input volume. (Hence the name: Convolutional Layer).
- Therefore, it is common to refer to the sets of weights as a filter (or a kernel), which is convolved with the input.
- The result of this convolution is an *activation map* (or feature map), and the activation maps for each different filter are stacked together along the depth dimension to produce the output volume.

CNN: Convolutional Layer

Example Filters



CNN: Convolutional Layer

Summary

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size W2×H2×D2 where:
  - $W_2=(W_1-F+2P)/S+1$
  - $H_2=(H_1-F+2P)/S+1$
  - $D_2=K$

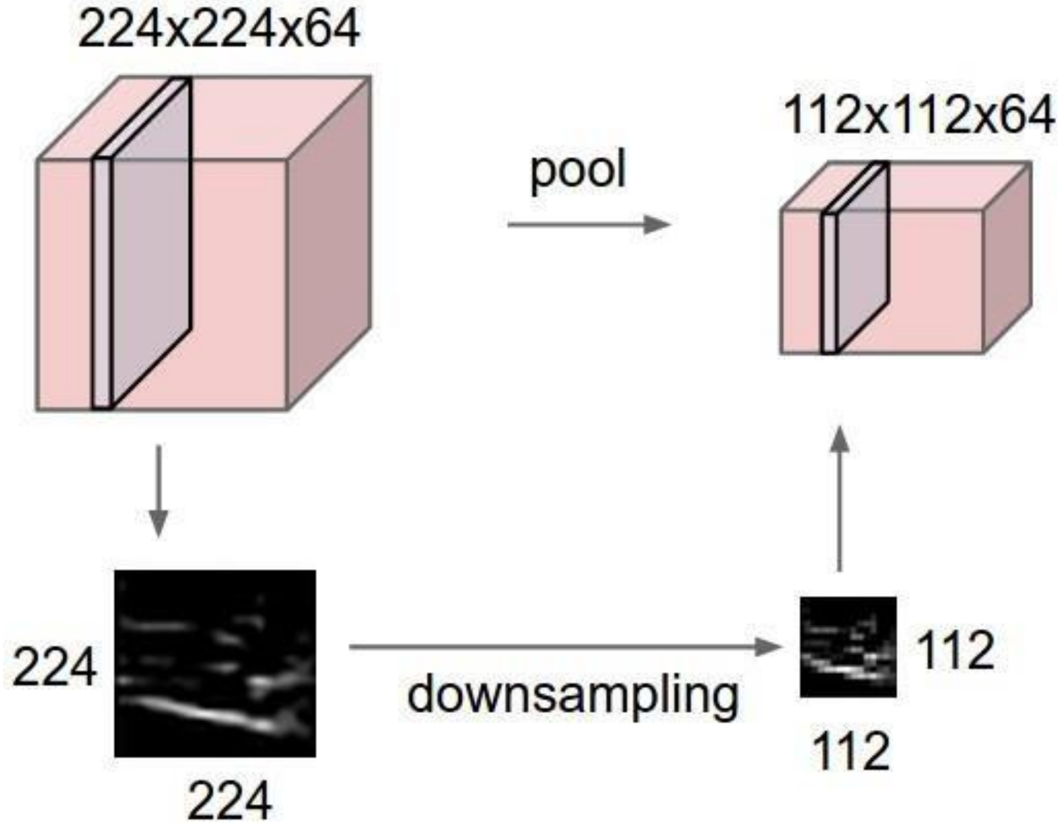
CNN: Convolutional Layer
Summary

- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

CNN: Pooling Layer

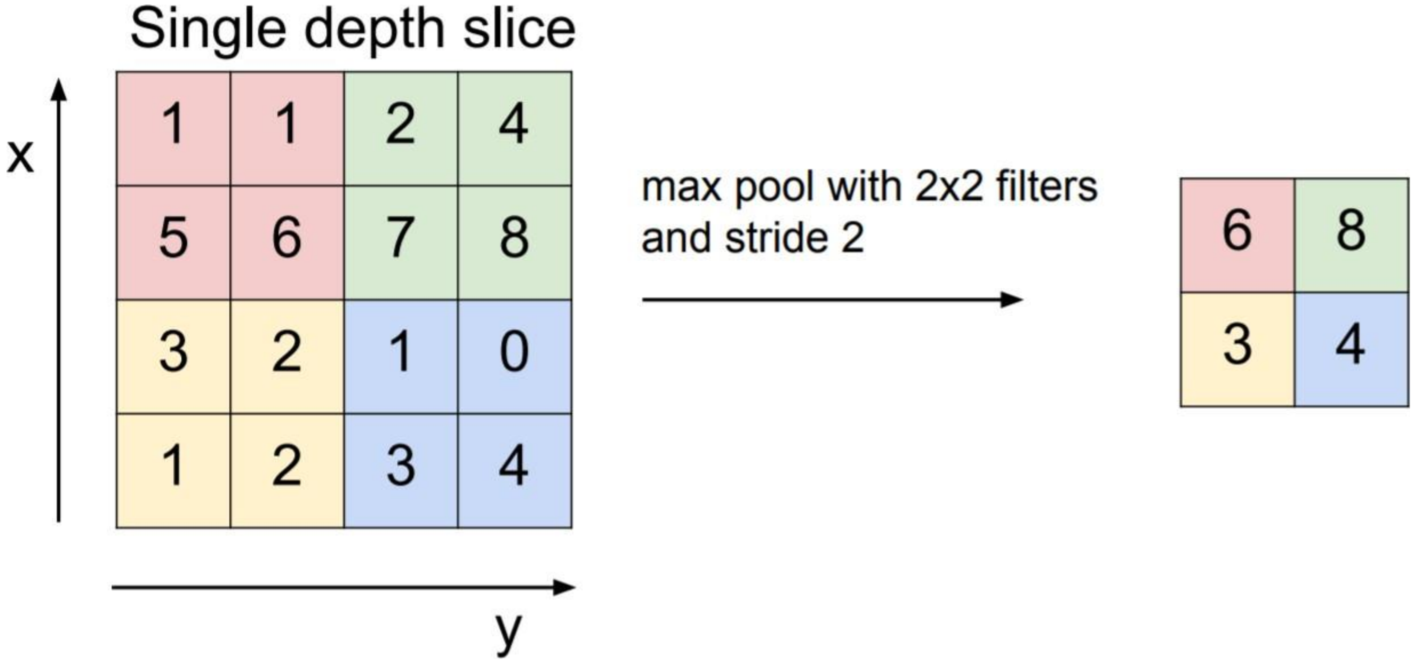
- The function of pooling layer
  - to progressively reduce the spatial size of the representation to reduce the number of parameters and computation in the network, and
  - hence to also control overfitting
- The Pooling Layer operates
  - independently on every depth slice of the input and resizes it spatially, typically using the MAX operation (ie: max pooling)
  - The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2, which downsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations

CNN: Pooling Layer



CNN: Pooling Layer

- Max pooling



Single depth slice

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

max pool with 2x2 filters and stride 2

6	8
3	4

CNN: Pooling Layer
Summary

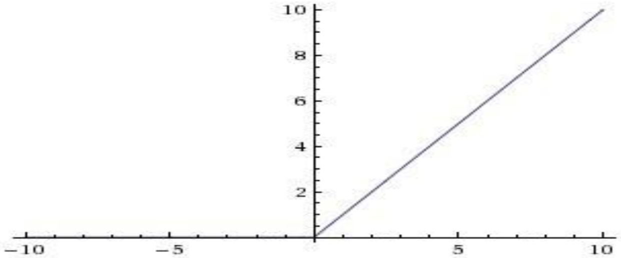
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$where:
  - $W_2=(W_1-F)/S+1$
  - $H_2=(H_1-F)/S+1$
  - $D_2=D_1$
- Introduces zero parameters since it computes a fixed function of the input

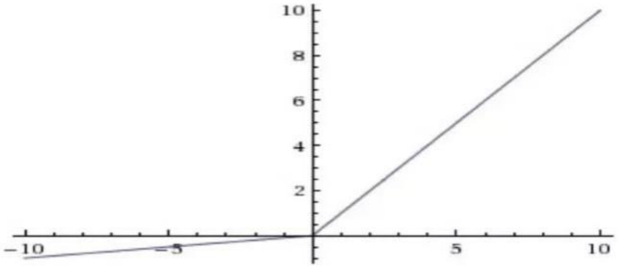
CNN: ReLU Layer

- Although ReLU (**Re**ctified **L**inear **U**nit) is considered as a layer, it is really an activation function:

  $$f(x) = max(0, x)$$

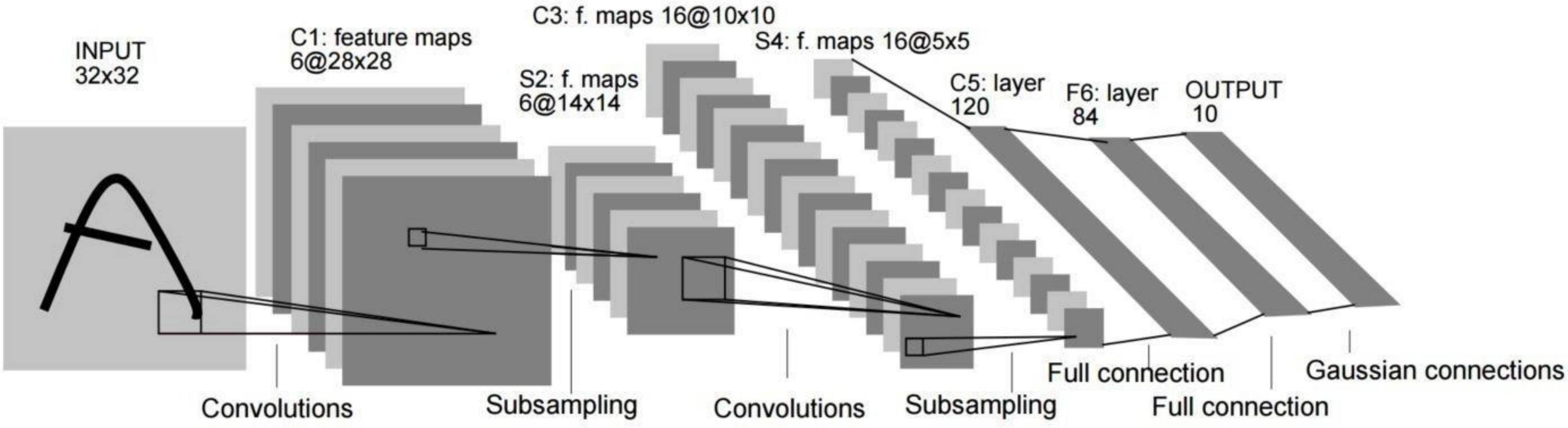
- This is favoured in deep learning as opposed to the traditional activation functions like Sigmoid or Tanh
  - To accelerate the convergence of stochastic gradient descent
  - Be computationally inexpensive compared to traditional ones
- However, ReLu units can be fragile during training and ‘die’. Leaky ReLUs were proposed to handle this problem.





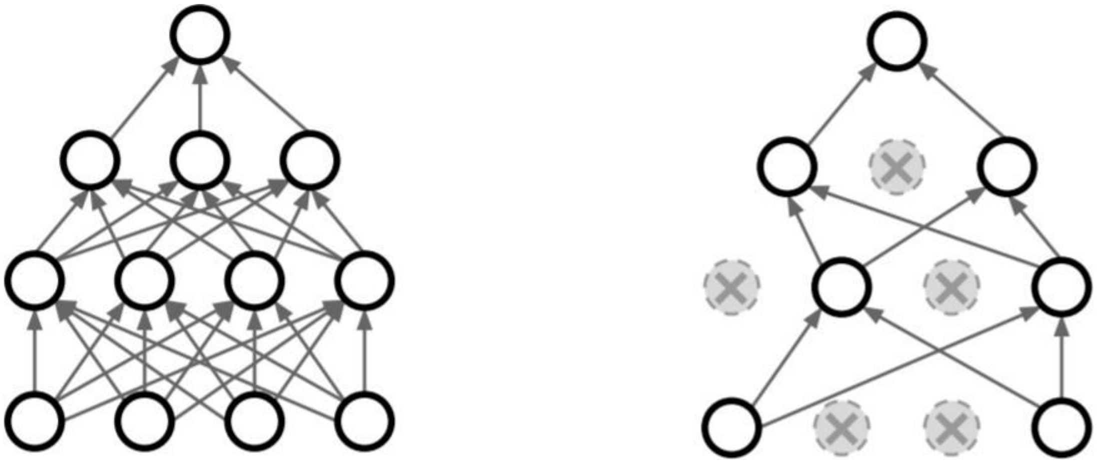
CNN: Fully-Connected Layer

- Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset.



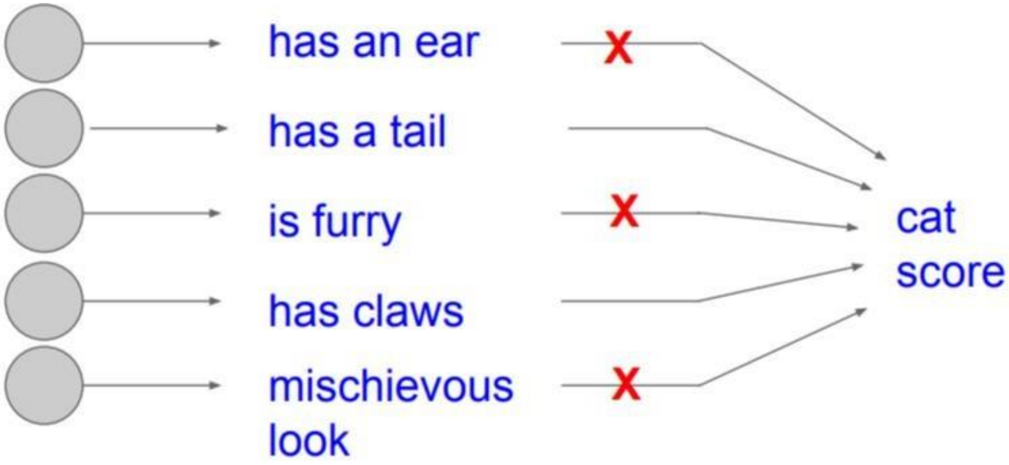
CNN: Dropout Layer

- Problem with overfitting – model performs well on training data but generalises poorly to testing data
- Dropout is a simple and effective method to reduce overfitting
- In each forward pass, randomly set some neurons to zero
- Probability of dropping is a hyperparameter, such as 0.5



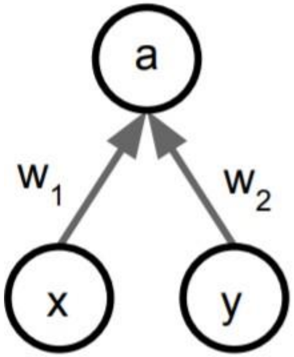
CNN: Dropout Layer

- Makes the training process noisy
- Forcing nodes within a layer to probabilistically take on more or less responsibility for the inputs
- Prevents co-adaptation of features and simulates a sparse activation
- Analogous to training a large ensemble of models but with much higher efficiency



CNN: Dropout Layer

- During test time, direct application would make the output random
- A simple approach: multiply the activation by dropout probability (e.g. 0.5)



Consider a single neuron.

At test time we have: $E[a] = w_1x + w_2y$

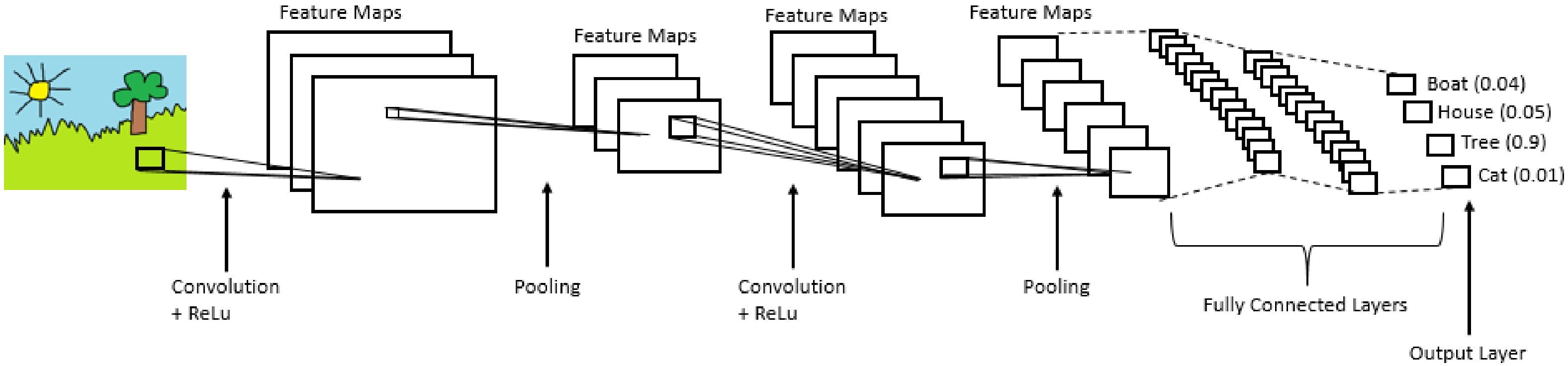
During training we have:

$$\begin{aligned}E[a] =& \frac{1}{4}(w_1x + w_2y) + \frac{1}{4}(w_1x + 0y)\\ &+ \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2y)\\ =& \frac{1}{2}(w_1x + w_2y)\end{aligned}$$

CNN: Output Layer

- The output layer produces the probability of each class given the input image
- This is the last layer containing the same number of neurons as the number of classes in the dataset
- The output of this layer passes through a Softmax activation function to normalize the outputs to a sum of one:

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$



CNN: Training

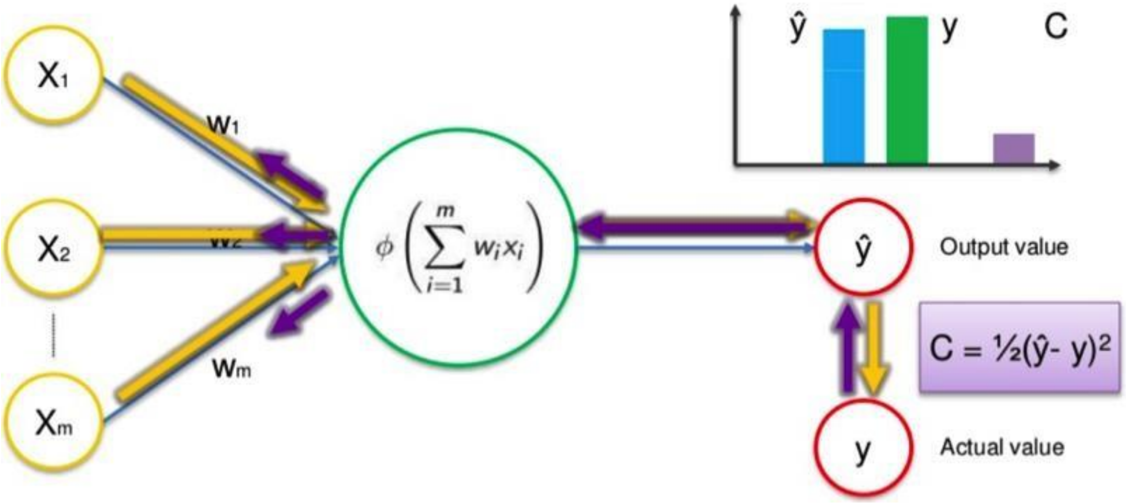
- Aloss function is used to compute the model’s prediction accuracyfrom the outputs
  - Most commonly used: categorical cross-entropy loss function

$$H(y, \hat{y}) = \sum_i y_i \log \frac{1}{\hat{y}_i} = -\sum_i y_i \log \hat{y}_i$$

- The training objective is to minimise this loss
- The loss guides the backpropagation process to train the CNN model
- Stochastic gradient descent and the Adam optimiser are commonly used algorithms for optimisation

CNN: Training

- Backpropagation in general:



https://www.superdatascience.com/blogs/artificial-neural-networks-backpropagation

1. Initialise the network.

2. Input the first observation.

3.Forward-propagation. From left to right the neurons are activated and the output value is produced.

4. Calculate the error in the outputs (loss function).

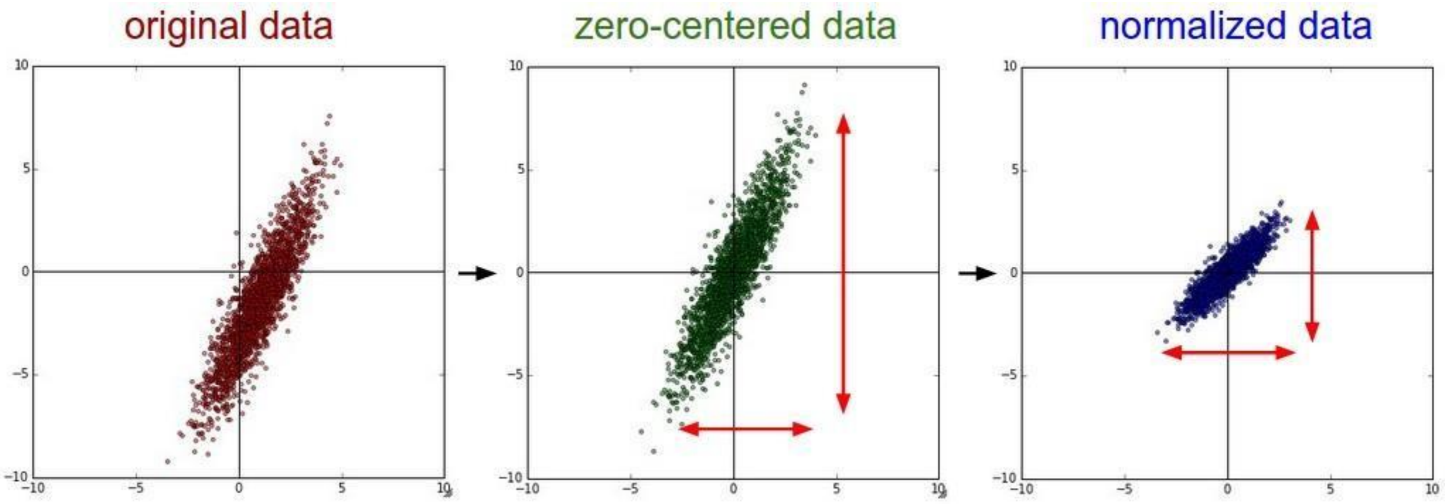
5.From right to left the generated error is back-propagated and accumulate the weight updates (partial derivatives).

6. Repeat steps 2-5 and adjust the weights after a batch of observations.

7.When the whole training set passes through the network, that makes an epoch. Redo more epochs.

CNN: Training

- Important details for training
  - Pre-processing: image scaling, zero mean, and normalisation

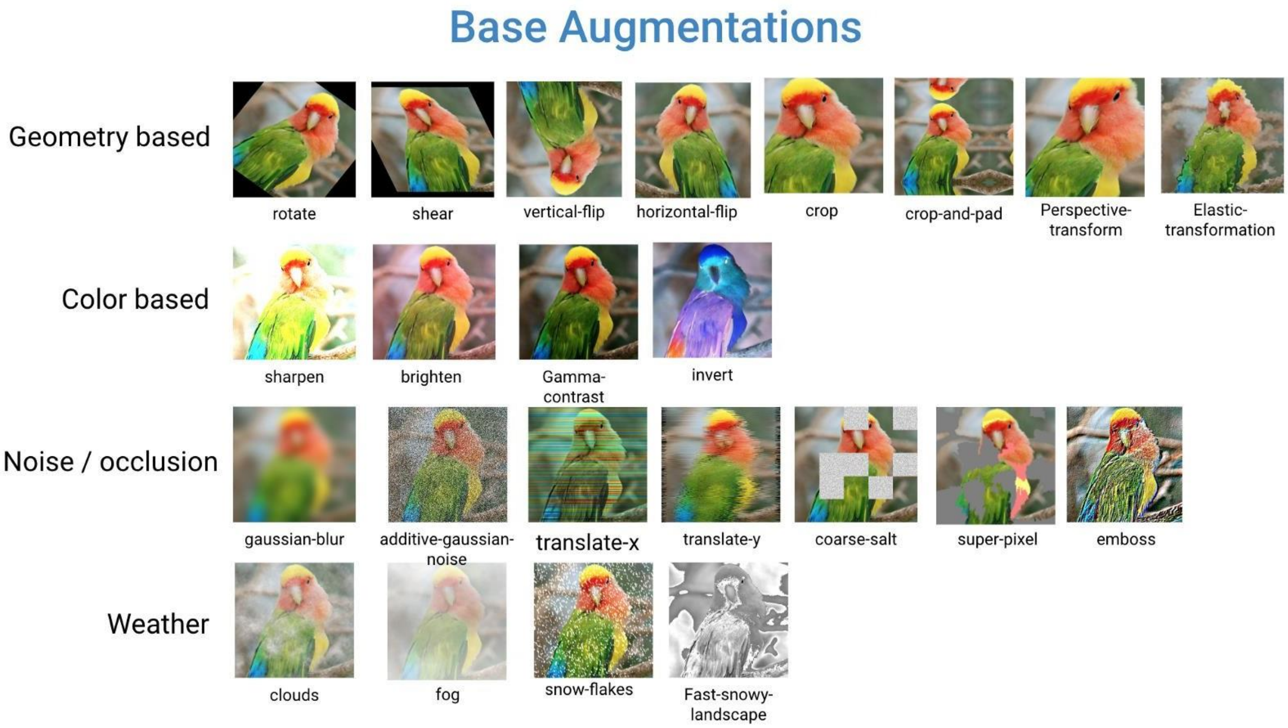


CNN: Training

- Important details for training
  - Data augmentation:
    - Essential for increasing the dataset size and avoiding over-fitting
    - More data augmentation often leads to better performance but also longer training time
    - Commonly used techniques include:
      - Horizontal / vertical flipping
      - Random cropping and scaling
      - Rotation
      - Gaussian filtering
    - During testing, average the results from multiple augmented input images

CNN: Training

- Important details for training
  - Data augmentation:
    - Need evaluation => not all techniques are useful



https://blog.insightdatascience.com/automl-for-data-augmentation-e87cf692c366

CNN: Training

- Important details for training
  - Weight initialisation
    - Cannot be all 0's => Need to ensure diversity in the filter weights
    - Use small random numbers => might aggravate the diminishing gradients problem
      - With calibration
      - Sparse initialisation
      - More advanced techniques
    - Use ImageNet pretrained models => not always possible

CNN: Training

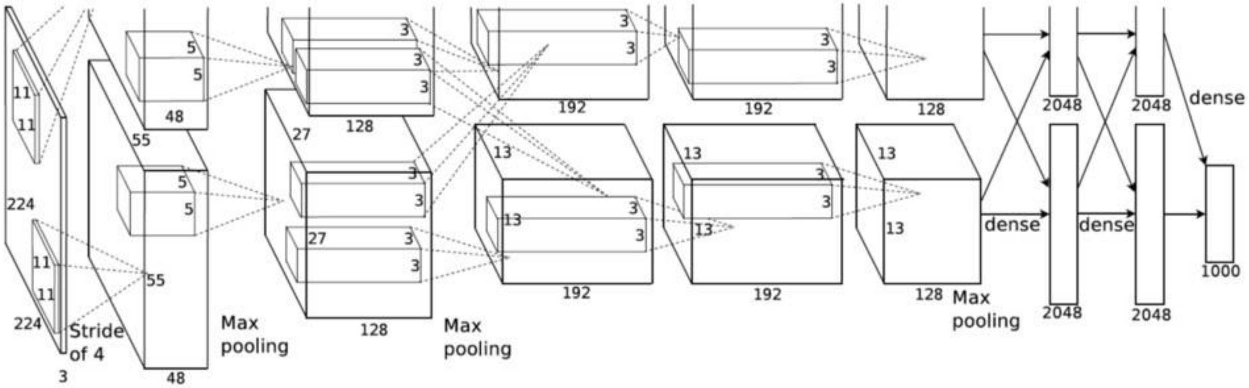
- Important details for training
  - Balanced training data
    - Important to have similar numbers of training images for different classes, so the optimisation would not be biased by one class
    - Use random sampling to achieve this effect during each epoch of training
    - Assign different weights in the loss function

$$\alpha_c = median\_freq/freq(c)$$

$$H(y, \hat{y}) = \sum_i y_i \log \frac{1}{\hat{y}_i} = -\sum_i y_i \log \hat{y}_i$$

CNN: Testing

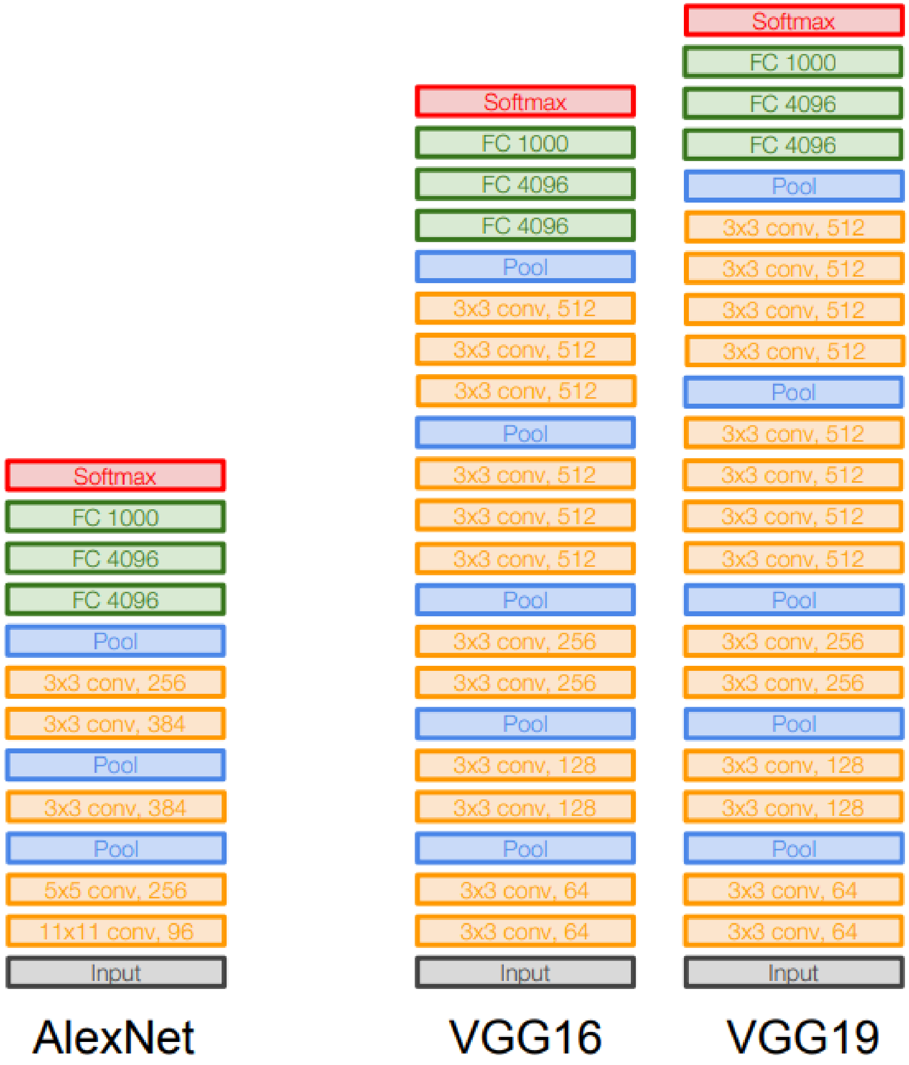
- Forward passes of the network throughout the layers give the prediction output of the input data



- Also related to – Transfer learning
  - CNN models trained on ImageNet can be applied to other types of images
  - It is possible to finetune only the last FC layers to better fit the model to the specific set of images
  - Especially useful for small datasets

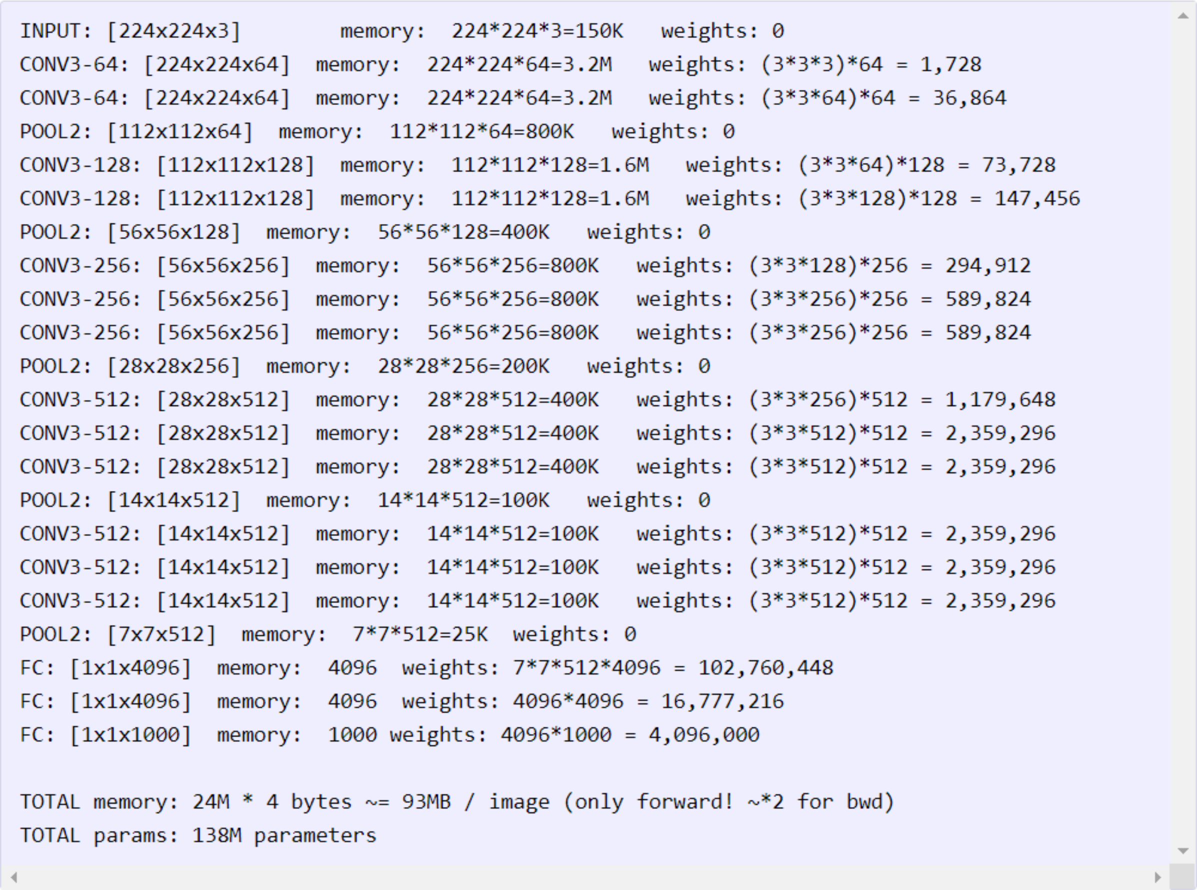
CNN: Model

- Case study: VGGNet



CNN: Model

- Case study: VGGNet

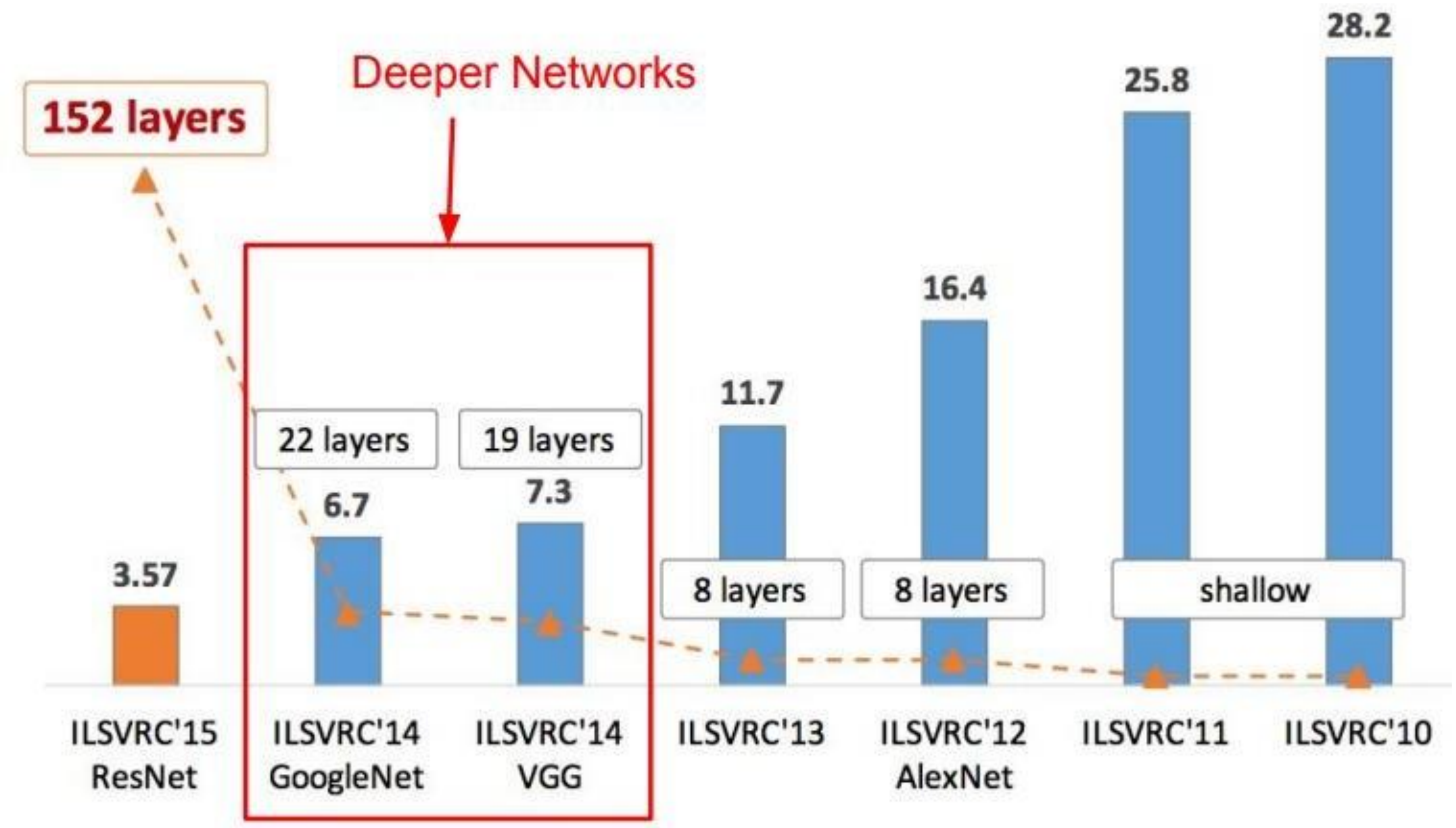


```
INPUT: [224x224x3]        memory:  224*224*3=150K   weights: 0
CONV3-64: [224x224x64]  memory:  224*224*64=3.2M   weights: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64]  memory:  224*224*64=3.2M   weights: (3*3*64)*64 = 36,864
POOL2: [112x112x64]  memory:  112*112*64=800K   weights: 0
CONV3-128: [112x112x128]  memory:  112*112*128=1.6M   weights: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128]  memory:  112*112*128=1.6M   weights: (3*3*128)*128 = 147,456
POOL2: [56x56x128]  memory:  56*56*128=400K   weights: 0
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*256)*256 = 589,824
POOL2: [28x28x256]  memory:  28*28*256=200K   weights: 0
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512]  memory:  14*14*512=100K   weights: 0
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512]  memory:  7*7*512=25K  weights: 0
FC: [1x1x4096]  memory:  4096  weights: 7*7*512*4096 = 102,760,448
FC: [1x1x4096]  memory:  4096  weights: 4096*4096 = 16,777,216
FC: [1x1x1000]  memory:  1000 weights: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters
```

CNN: Model

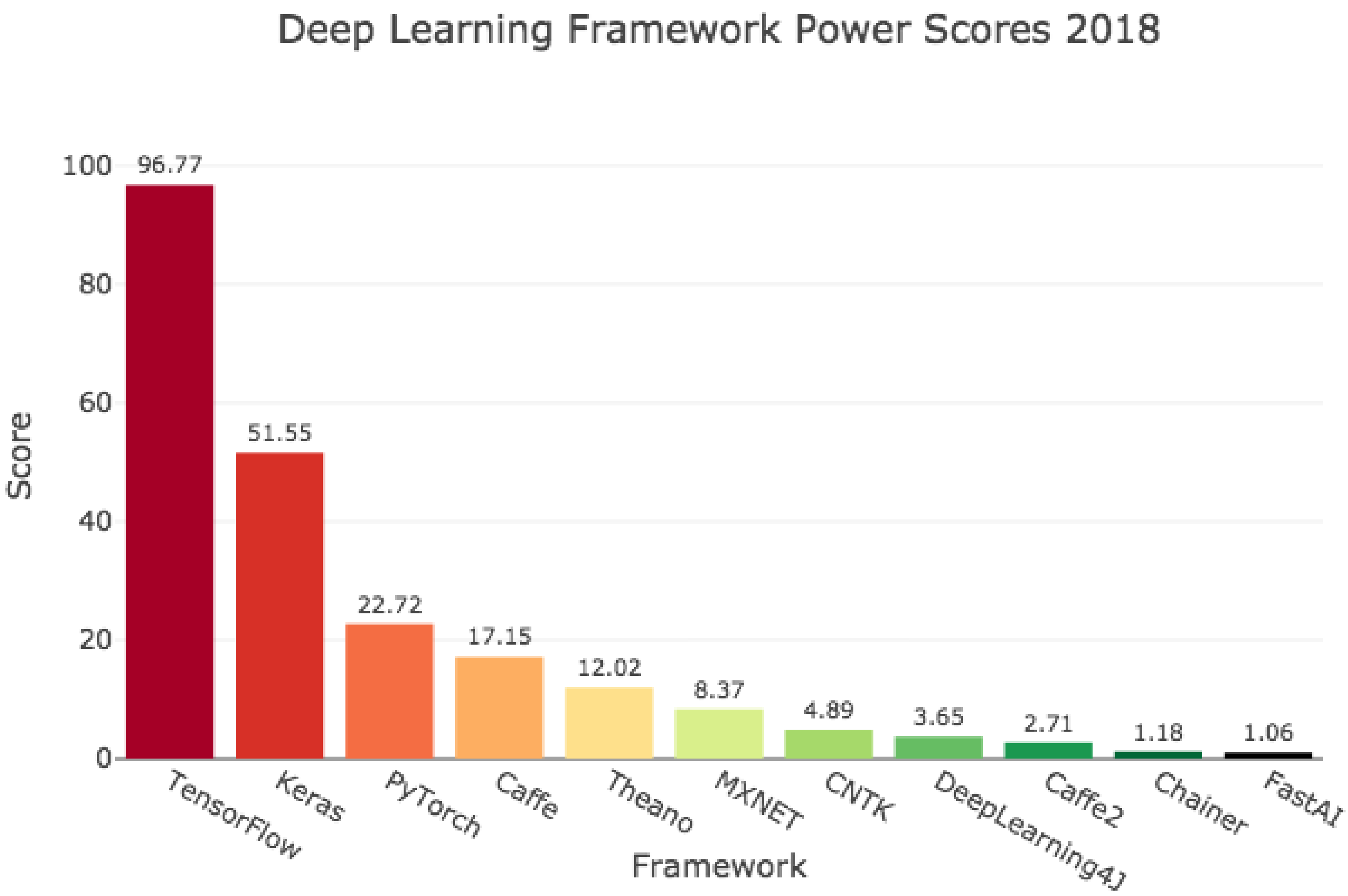
- Case study: VGGNet



Well-known Models

- Object Recognition
  - AlexNet (2012)
  - GoogLeNet
  - VGGNet
  - ResNet
  - Inception v3/v4
  - DenseNets is the current state-of-the-art
- Semantic Segmentation
  - Multi-scale CNN (2012)
  - FCN
  - U-net / V-net
  - U-net / V-net with skip/dense connections
  - Many other variations
- Adversarial
  - Generative Adversarial Networks (2014)
  - Pixel2pixelGans
  - CycleGans

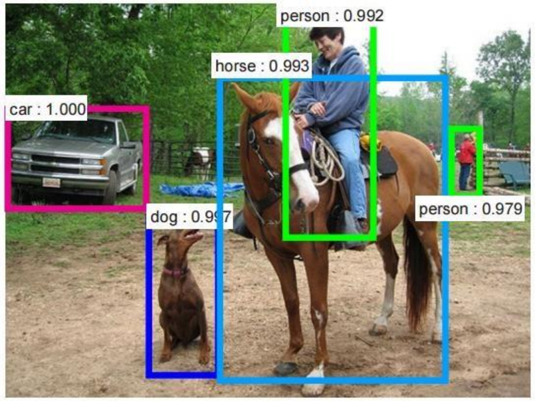
DL Frameworks

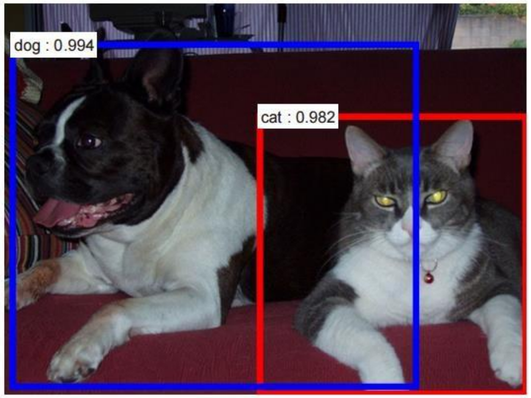


https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a

Object Recognition

- R-CNN, Fast R-CNN, Faster R-CNN, Mask R-CNN
- YOLO
- R-FCN
- SSD

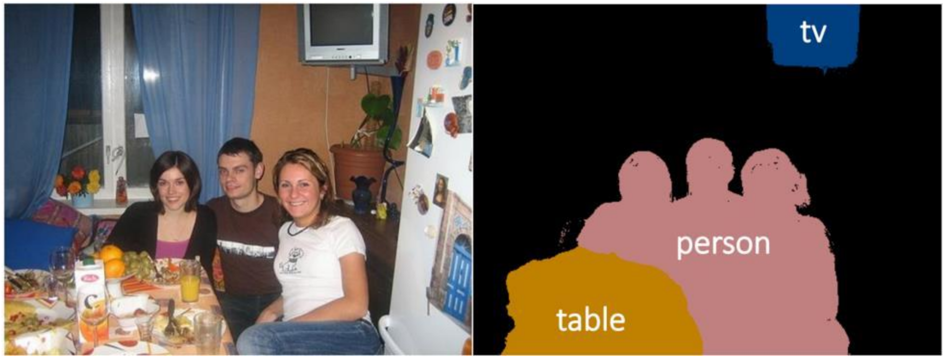




References and further reading: https://github.com/kjw0612/awesome-deep-vision

Semantic Segmentation

- Deep Parsing Networks
- BoxSup
- Fully Convolutional Networks for Semantic Segmentation
- SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation
- Fusing the Boundaries of Boundary Detection Using deep Learning



References and further reading: https://github.com/kjw0612/awesome-deep-vision

Scene Understanding

- Explain Images with Multimodal Recurrent Neural Networks
- Unifying Visual-Semantic Embeddings with Multimodal Neural Language Models
- Deep Visual-Semantic Alignments for Generating Image Description
- Learning Query and Image Similarities with Ranking Canonical Correlation Analysis

man in black shirt is playing guitar.

construction worker in orange safety vest is working on road.

two young girls are playing with lego toy.

boy is doing backflip on wakeboard.

References and further reading: https://github.com/kjw0612/awesome-deep-vision

References

- Some slides were adopted from the class notes slides from Stanford course cs231n
- Gradient-based learning applied to document recognition, Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner
- A fast learning algorithm for deep belief nets, Hinton, G. E., Osindero, S. and Teh, Y.
- ImageNet Classification with Deep Convolutional Neural Networks, Alex Krizhevsky and Sutskever, Ilya and Hinton, Geoffrey E
- Visualizing and Understanding Convolutional Networks, Zeiler, Matthew D. and Fergus, Rob
- Very Deep Convolutional Networks for Large-Scale Image Recognition, Karen Simonyan, Andrew Zisserman
- U-Net: Convolutional Networks for Biomedical Image Segmentation, Olaf Ronneberger, Philipp Fischer, Thomas Brox
- Going deeper with convolutions, C. Szegedy et al.
- Generative Adversarial Nets, Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio
- Deep Residual Learning for Image Recognition, K. He, X. Zhang, S. Ren and J. Sun
- Densely Connected Convolutional Networks, Gao Huang, Zhuang Liu, Kilian Q. Weinberger